Interface

2003 May

［表紙デザイン：㈱プランニング・ロケッツ］

Contents 2003 May

特集

オブジェクト指向の導入で開発効率向上！

Contents 2003 May

話題のテクノロジ解説

ショウレポート&コラム

一般解説&連載

情報のページ

エンタープライズITソリューション展

NET & COM 2003

北村俊之

「実践!ブロードバンドで企業革新」をテーマに「NET&COM 2003」が2月5日(水)～7日(金)の3日間，幕張メッセで開催された．主催は日経BP社．本展示会は今年で11回目の開催となり，10周年を迎えた．展示会全体を「ネットソリューション」,「システムソリューション」,「ネットセキュリティ」,「Webソリューション」,「CT/CRMソリューション」の五つの展示ゾーンに分け，企業の情報・ネットワークシステムのソリューション専門展という位置づけを前面に押し出す形となっていた．また，展示会と併催されていたセミナーでは，スペシャルセッションや特別講演，トップカンファレンスのほか，ITプロフェッショナル向けの専門セミナー，3トラック17セッションなど，80を超えるセミナーが開催されていた．最終的な来場者数は，75,328人となっていた．

● ネットセキュリティ

このゾーンでは，トレンドマイクロ，シマンテック，ネクサンティス，フェニックステクノロジーズなどが軒を並べており，各ブースともに盛況であった．トレンドマイクロでは，ウィルスの大規模感染の予防から復旧の事後措置までをトータルに管理する「アウトブレークライフサイクルマネジメント」と，それを実現するトレンドマイクロEPS(エンタープライズプロテクションストラテジー)に対応する製品群のデモを行っていた(**写真1**).

〔**写真1**〕**トレンドマイクロのブース**

ネクサンティスでは，セキュリティとスマートカードにフォーカスしたソリューションを展示しており，「AccessMaster」,「Portal Xpert」,「SecureScan」,「Desktop Solution」などの多彩なセキュリティソリューションで来場者の注目を集めていた(**写真2**)．シマンテックでは，侵入検知製品，セキュリティポリシー監査製品，独自の統合型セキュリティ製品およびセキュリティ全体のマネジメントツールの展示を行っており，こちらも来場者の関心が高いブースだった．

〔**写真2**〕**ネクサンティスの展示**

● ネットソリューション

セイコーインスツルメンツでは，IP-VPNや広域Ethernetなどの通信サービスに対応したブロードバンド製品「BlueBrick」,「EXAtrax」や「NSシリーズ」新製品の展示を行っていた(**写真3**)．コンテックでは，54Mbps無線LAN「FLEXLAN DS540シリーズ」の出展を行っていた．同シリーズの特徴は，5GHz帯と2.4GHz帯のデュアルバンド対応で，ルータをこえたローミングを可能にする独自のIPトンネル機能，IEEE802.1x対応と独自暗号搭載によるセキュリティをもつなどで，ビジネスユーザーやSI業者から高い評価を得ているという．今春発売のアンテナ一体型マイクロアクセスポイントの出展も行われていた．岩崎通信機では，Voiceアプリケーションと自社開発のSIPテクノロジをコアとしたソリューションの展示を行っていた．

〔**写真3**〕**SIIのATM-Ethernetスイッチ「EXAtrax」**

クレオでは，画像，音声，データによる遠隔会議を実現するインターネット会議システム「フェース・カンファレンス」シリーズの紹介を行っていた．こちらは，パソコンおよびブロードバンドネットワーク環境に，標準的なPCカメラとヘッドセットを用意するだけで，遠隔地とビジュアル会議が開けるというものである．MPEG-4準拠のビデオCODECを採用することで，ネットワークにかかる負荷を抑えつつ，高品質な画像処理と音声処理を実現している．

● システムソリューション

サイボウズでは，大規模環境で利用可能な「サイボウズガルーン」を操作画面や事例などを交えてデモを行っていた．オービックビジネスコンサルタントでは，ブロードバンド環境での運用に最適化した「奉行BroadBand Edition」をデモを交えて紹介しており，和風のブースとあいまって来場者の注目を集めていた(**写真4**).

〔**写真4**〕**オービックビジネスコンサルタントのブース**

SRAでは，オープンソースデータベースである「PostgreSQL」のWindows版「PowerGres」の展示を行っていた．同製品は，既存の商用データベースと比較して遜色のない性能，コストパフォーマンスで定評がある(**写真5**)．コネクティクスでは，「Virtual PC」および「Virtual Server」の展示を行っていた．「Virtual Server」はインテルベースの大型サーバを物理サーバに統合可能にするWindowsベースのサーバアプリケーションとのことである．インサイトテクノロジーでは，Oracleのパフォーマンス管理，運用監視ツール「Peformance Insight」やデータベース診断サービス「Au-DBit」の紹介を行っていた．「Peformance Insight」は，今後WebサーバやSAP/R3など，システム全般への対応を予定しているとのことであった．

〔**写真5**〕**SRAのPowerGres**

● Webソリューション

エンピレックスでは，Webおよびネットワークアプリケーションのための統合管理ソリューションを展示していた．「e-TEST Suite」は，Webアプリケーションの機能性，拡張性，可用性をテストする統合テストツール．「OneSight」は，ネットワークコンポーネントからアプリケーションのパフォーマンスまでを監視するツールである．「Hammer PacketSphere」は，全二重ギガビットのワイヤ速度まで対応可能なIPネットワークエミュレータとのことだった．最近は，発信側もネットパフォーマンスを気にするようになり，こうした管理ツールやエミュレータへの注目が高まっているとのことだった．

フォトロンでは，映像ナレッジマネージメントシステム「Power Index」の展示を行っていた．動画コンテンツをメタ情報，テキストベースでディジタル検索，管理するシステムで，エンコーダ，サーバ，ビューワの三つのユニットで構成されている(**写真6**).

〔**写真6**〕**フォトロンのブース**

Show report

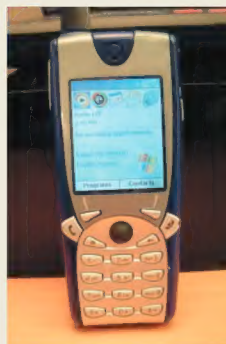
The Microsoft Windows Embedded Developers Conference 2003 Japan

■日時：2003年2月18日(水)～19日(木)
■場所：赤坂プリンスホテル(東京都千代田区)

マイクロソフトによる組み込みシステムに関するカンファレンスが開催された．キーノートスピーチでは，Embedded and Appliance Platform Groupのジェネラルマネージャである Todd Warren氏により，同社が組み込みシステムにおいて実施しているWindows Embedded Partnerプログラムを強化し，パートナー間の連携によるビジネスの拡大などを視野に入れたパートナー"ECO(循環)システム"を構築することなどが発表された．また，壇上ではWindows CEなどを採用した組み込み機器が紹介された．

続いてインテルによるGeneral Sessionでは，同社取締役通信事業部本部長の高橋恒雄氏により，ワイヤレスMMX技術が紹介された．これはXScaleに64ビットSIMDアーキテクチャを拡張し，マルチメディア用途などでのパフォーマンスを向上させるものである．通常は機能拡張による消費電力増加が問題になるが，ワイヤレスMMXでは，機能向上と低消費電力化を両立しているとのことである．

そのほかに行われたセッションは「Windows CE .NET Platform Builder概要」，「Windows CE .NETオペレーティングシステムおよびカーネルアーキテクチャ」，「.NET Compact Framework概要」，「Visual Studio .NETによるデバイスアプリケーション開発」など．

併設された展示会では，インテルのXScaleを用いた携帯電話端末「Windows Powered Smartphoneコンセプト・デザイン」などが展示されていた．

Windows Powered Smartphoneコンセプト・デザイン

Windowsを採用した計測機器

Todd Warren氏

Windowsで動作するミシン

高橋恒雄氏

GAIA System Solutions Inc. & ELT,Inc Premium Conference

■日時：2003年2月19日(水)
■場所：TATOU TOKYO(東京都港区)

GAIAとELTによるプレミアムカンファレンスが開催された．

トヨタ自動車第5開発センター第2電子技術部部長の重松崇氏による「自動車制御用組み込みソフトウェアの品質管理」では，自動車用組み込みソフトウェアにおいて，近年ソフトウェアの開発料が急増している現状を述べ，それに対して，品質管理のために「構造設計と部品共有化」，「開発プロセスの標準化」，「品質の推定方法」，「実機テストの評価環境」などが行われているとのことだった．また，今後のソフトウェア開発プロセスにおいて，オブジェクト指向技術による部品化やUMLの導入によるシステム記述のポータビリティの向上が重要であることなどが語られた．

また，ソニー・コンピュータエンタテインメント常務兼CTOの岡本伸一氏による「コンカレント開発とシステムシミュレーション」では，PlayStation2の例をあげ，システム開発サイクルの短縮化のためにシミュレータを用いた開発が有効であることなどが語られた．

そのほかの講演は，GAIAの販売するシステムシミュレータ「CoMET/METore」を開発したVaST Systems Technology社のCEOであるGraham Hellstrand氏による「米国電装メーカーにおける成功事例」，ソニーコアテクノロジー＆ネットワークカンパニーメモリースティック事業部の寺戸郁夫氏による「メモリースティックの広がる世界とデバイスドライバーの開発～イーエルティとのJoint Development～」，東京大学坂村健氏による「ユビキタスコンピューティングとT-Engine」など．

重松　崇氏

岡本伸一氏

会場のようす

情報家電のリテラシー

山本　強

昨年あたりから，DVDとそれに関連した家電製品や情報機器が話題になっている．出荷額の統計によると，昨年はDVDプレーヤの出荷量がVCR（ビデオカセットレコーダ）を越えたのだとか．確かに，家電量販店の売り場を見ると，VCRよりもDVDプレーヤやDVDレコーダのほうが目立っているように感じる．アナログレコードからCDへ移行したときの経験から考えると，ビデオカセットがDVD-RやDVD-RWなどに駆逐されるのは時間の問題だろうと思う．

そんな折，我が家でもCATVで映画チャネルが見られるようになった．そこそこに映画マニアでもある筆者は，これをきっかけにHDDレコーダが欲しくなり，久々に家電量販店の売り場を物色することとなった．映画が好きとはいってもAVマニアというほどでもない筆者は，ともかく安いDVD-Rレコーダはないかと探すことにした．そうして，60GバイトHDD＋DVD-R/RAMのレコーダが8万円台で売られているのを見つけ，思わず衝動買いをした．

じつのところ筆者は10年ほど前に研究目的で，30分だけ録画できるアナログのLDレコーダを買ったことがあるが，その当時の値段は350万円だったのである．IT分野の技術革新の凄まじさを感じさせてくれる商品である．なかなか凄いスペックの家電製品で，使うのに相当なリテラシーが必要であるように感じた．

複雑な情報家電の接続モデル

これを買った目的がCATVからの録画だったので，まずCATVのセットトップボックス（STB）につなぐことになるのだが，どのようにつなぐのが正しいのかがわからない．最近のAV機器は必ず複数の入出力端子を備えているので，その可能な接続の組み合わせは，ちょっと大げさにいうと機器の数をNとすればN乗のオーダということになる．TV＋レコーダの単純な組み合わせなら悩むことはないのだが，STBが絡んでくると話がやっかいになる．

などと，ぶつぶついいながら信号の流れを整理してつなぎ込んだが，ラックの後はAVケーブルによってくもの巣状態になってしまった．マニュアルには代表的な接続例が書かれているのだが，コンポーネント化されたAV環境はいろいろなケースがあり，マニュアルに書かれているものと同じとは限らない．難しいといわれるPCのセットアップのほうがよほど標準化されているように思う．

さらに，最新のHDDレコーダには100Base-Tのコネクタが付いていて，PCとの連携も可能になっている．これも大変な話で，そこまで考えるAVマニアのPCにはきっとMPEG-2のエンコード/デコードのボードも装着されているはずである．したがって，AV接続の複雑さはさらにもう一段上がることになる．こんな高度なリテラシーを要求する家電製品がヒット商品になるとは，日本国民は相当にITリテラシーが高いと自慢してよいのではないだろうか．

さらに高度なリモコンリテラシー

何とかセットアップが終わり，スイッチを入れてようやく画面が現れるのだが，じつはその先にまた難題が控えていた．あまりに機能が多いのである．HDDの容量が60Gバイトあるということは，DVD換算で15枚以上記録できるわけなので，タイトルなどを見るためにブラウザが必要になる．

録画する方法だって，記録先がHDD，DVD-RAM，DVD-Rと3パターンもあるし，録画ソースもTV，CATV，内部ダビングと複雑怪奇である．そのうえ，記録フォーマットが4通りくらいある．これをすべて理解して使い分けるのはなかなか大変である．少なくとも，メールやWWWをアクセスするための情報リテラシーより高度な知識を求められているように感じる．そして，これらを操作するのがリモコンという機種依存の端末装置ということになるわけだ．

リモコンは大きさが限定されるので，ボタンの数に限りがある．限られたボタンで再生ブラウザから編集，ダビングまで行うわけだから，操作シーケンスが深くなるのである．メニューになっていればよいというものではない．メニュー階層が3段階を超えたとたんに，何をやっているのかわからなくなる．いちばん困るのが，リモコンの形やボタン配置が機種ごとに違うことである．難しいといわれるPCのキーボードだって，キー配置は基本的に同じなのである．ボタンが少ないから簡単だというのは大きな誤解である．

情報リテラシーってなんだろう

HDD付きDVDレコーダの中身は，冷静に考えれば高性能PCにDVD-RとMPEG-2付きTVチューナをつないで，箱をDVDプレーヤのようにデザインしたものである．すなわち，中身はPCだが箱が家電製品ということである．だから使い方もPC的になり，情報リテラシーならぬ家電リテラシーが必要になったということだろう．

これからは情報リテラシーが重要だといい始めたのはもう何年も前の話である．e-Japan戦略の効果もあってか，学校内インターネットからIT講習会まで，いたるところで情報リテラシー教育を受けることができるようになった．しかし，構造も機能も最上位レベルのPCに匹敵する情報家電の使い方を学習するための講習会というのは聞いたことがない．

人間，本当に欲しいものや，なければ生活に支障が出るようなリテラシーは自然に身につくものだということを，このHDDレコーダは証明しているのではないだろうか．

やまもと・つよし　北海道大学大学院工学研究科電子情報工学専攻
計算機情報通信工学講座　超集積計算システム工学分野

ハッカーの常識的見聞録

29

広畑由紀夫

今月の常識

Windows Server 2003 RC2 評価版に見る進歩

☆MSDN会員以外でも入手が可能な「Windows Server 2003 RC2日本語版」の出荷が2003年2月より開始されました．今回は評価版ではありますが，その性能の一端を見てみましょう．

マイクロソフトの次期サーバ製品「Windows Server 2003」の発売が延期を繰り返し，待ち望まれているにもかかわらず，いまだに製品版の出荷は行われていません．しかしながら，ようやく評価版としてインストール後360日間利用できるWindows Server 2003 Enterprise版が提供されたので，さっそく導入テストを行ってみました．

さて，Windows XP Home/Professionalを新規インストールされた方はご存じだと思いますが，NTFSの簡易フォーマットがサポートされているため，HDDの初期化に要する時間が非常に短縮されています．評価のためだけにハードディスクのフォーマットに何時間もかけるという無駄が避けられます．

● 利用するサービスの選択

Windows Server 2003を導入した直後にまず設定を行うのが，「サーバの役割管理」です．これは，導入するサーバをどのように使うかといった基本的な設定で，ファイルサーバとして使うのかどうかといった設定を行います．そして，必要なサービスについては別途インストールを行います．

以前は，IISなど標準的なサービスと思われるコンポーネントがてんこ盛りだったこともありましたが，今回はサーバとして動作させる最低限というよりも，管理者用クライアントとして最低限のコンポーネント程度しかインストールされないようです．スタンダード版ではファイルサーバとしての機能を追加できるようになっており，簡単な導入テストとしては十分でしょう．

● Windows Server 2003での開発環境テスト

Visual Studio.NET 2003の評価版をすでにWindows XPなどでテスト/評価されている方は，新規インストールで導入できるようです．筆者のテスト環境でも，日本語版の評価用キットでインストールならびにC++クラスライブラリの動作確認などを開始しています．

● クライアントとしての評価版Server

本来はサーバとして安定した動作を期待して導入するはずです．しかしながら，筆者としてはクライアントとしての安定度やデバイスドライバなどの互換性も確認してみたくなりました．サーバ用クライアントとして最低限のコンポーネントをインストールした状態でサービスの開始を見てみましたが，スタンダード版をまず導入してみるのが無難でしょう．

筆者のテスト環境では，すでに多くのコンポーネントが非公式ながらも導入されていたため，標準で添付されているデバイスドライバやアプリケーションインストールCDからのハードウェア専用デバイスドライバおよびツールのインストールが可能でしたが，周辺機器によっては対応していないものもあると思います．

〔図〕Windows Server 2003情報サイト・早期評価プログラムの入手先
（http://www.microsoft.com/japan/partner/products/windowsfamily/windowsserver2003/default.asp）

● 使用感と製品への期待

クライアントとしてテストする際に欠かせないDirectX9やWindows Media Player9についても，ダウンロードしてインストールすることができました．また，Windows Updateによる自動アップデートに関しても，すでにテストが可能になっているように，360日間の期間の限定版とはいえ，かなりのところまで導入に関する問題点を洗い出すための評価が可能になっています．

試用した感じでは，すぐにでも通常のデスクトップとして使いたいほどのチューニングがなされていて，デバイスドライバなどの相性しだいでは，それなりの反応や安定度も得られるようです．筆者の環境では，Direct3Dのフル機能がソフトウェアテストでは通りましたから，3Dに特化した市販ゲームソフトさえも動作するようです．ライセンス料しだいでは，サーバ機能をインストールせずに個人の端末で使用してみたいと考えています．

皆さんも試用してみませんか．筆者は製品版まで「試用」という名目で使っていくつもりです．

ひろはた・ゆきお　OpenLab.

特集 オブジェクト指向の導入で開発効率向上！

うまくいく！組み込み機器の開発手法

組み込みシステム開発の現場では何が問題で，それに対してどんな解決策があるのかを解説する．まず現場エンジニアの立場から，例を示しながら，効率的なオブジェクト指向の導入法を解説する．エレベータと電波時計システムを例に，UMLを用いて，Model Driven Architecture/ユースケース駆動型という異なるモデリングを行う．実装コードも紹介し，全ソースは本誌付属CD-ROM InterGiga No.30に収録する．

ソフトウェア開発を行う組織を指導・支援するソフトウェア開発コンサルタントという仕事がある．従来はオブジェクト指向やUMLを実際の開発に適用するための指導・支援を行ってきたが，最近では，開発全体の課題やビジネスゴール，事業環境，社会環境といった全体の最適化を支援するようになってきた．このソフトウェア開発コンサルタントのうち，組み込みソフトウェア開発をおもな対象とするグループによる，組み込みソフトウェア開発の生産性/品質を向上するための方法論を，技術面，組織面，人的な面などにわけ，特集後半で解説する．

Prologue

組み込み機器開発における問題点と解決手法

井上　樹/川口　晃/佐藤啓太/杉浦英樹/橋本隆成

はじめに

本特集は，「今，組み込みソフトウェアで何が問題になっていて，それに対してどのようなソリューションがあるのか」をコンセプトにまとめている．前半部分は，開発の第一線で活躍しているエンジニアから，現場での活動についてたいへん力の入った報告を行う．

後半部分では，組み込みソフトウェア開発のコンサルタントから，また現場からは一歩下がった視点での提言を行っている．最後に，両者の意見をすり合わせ，問題意識が一致している点，ずれている点，今後の取り組みなどをまとめる．

図1に，今回の特集のコンセプトを示す．

1 構成

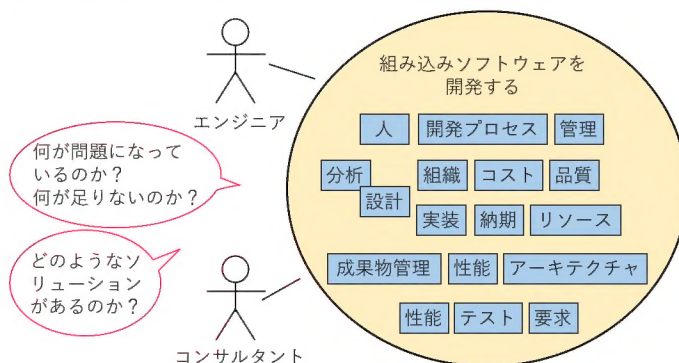
1.1 現場のエンジニアからの報告

まず，組み込みソフトウェア開発技術の最前線で苦闘しているエンジニアから，「組み込みソフトの開発を考えたときに世の中に不足していること」をテーマにホットな報告がなされる．

● **事例**

具体的な題材として，エレベータと電波時計という2種類の開発事例が紹介される．この二つの事例は，両方ともUMLを用いてモデリングされているが，それぞれMDA（Model Driven Architecture）的なソリューションを指向したモデルと，よりプロセス重視で属人性を排除しようとしたモデルになっている．

このように異なるモデリングのスタイルを同時に比較できるのはまれであり，貴重な事例といえる．しかもソースコードまで添付されている（一部掲載．全ソースは本号付属CD-ROM InterGiga No.30に収録）．UMLモデルがどのようにソースコードに変換されるのかを確認するのには良い機会だと思う．

〔図1〕特集のコンセプト

● **事業としての組み込みソフトウェア（開発現場で考えるべきこと）**

事業（ビジネス）の視点で組み込みソフトウェアを見たとき，開発現場が考えるべきことが報告される．これまで，組み込みソフトウェア関係の書籍や雑誌などでは，あまり触れられなかった部分である．後述するように，そう遠くない将来，組み込みソフトウェアの事業性が検討されることもあると思う．今後，欠かすことのできない視点である．

● **組み込みとオブジェクト指向**

なぜ，組み込みでオブジェクト指向が広まらないのかをテーマに，現場のエンジニアのリアルな視点でオブジェクト指向が語られる．

1.2 コンサルタントからの報告

ここでは，組み込みソフトウェアのコンサルタントが，現場から一歩下がった冷静な視点で，事業としての組み込みソフトウェア開発について，課題の体系化と取り組むべきポイントの提言を行う．

● **課題の体系化**

まず，技術，社会，経済，政治・法律の視点で，事業としての組み込みソフトウェアの環境を分析し，解決すべき課題を整理，体系化する．

● **技術面の取り組みポイント**

組み込み系と業務系のソリューションの違いを整理し，組み込みソフトウェアへのプロダクトラインの有効性を示す．さらにプロセス，仕様，ツール，構成管理などの技術的側面を整理する．また，ノウハウの外在化の重要性について述べる．

● **組織・ビジネス面の取り組みポイント**

事業としての組み込みソフトウェアを考えるうえで不可欠な企業人としての視点，すなわち競争優位性，組織マネジメント，暗黙知と形式知のスパイラルなどについて述べる．

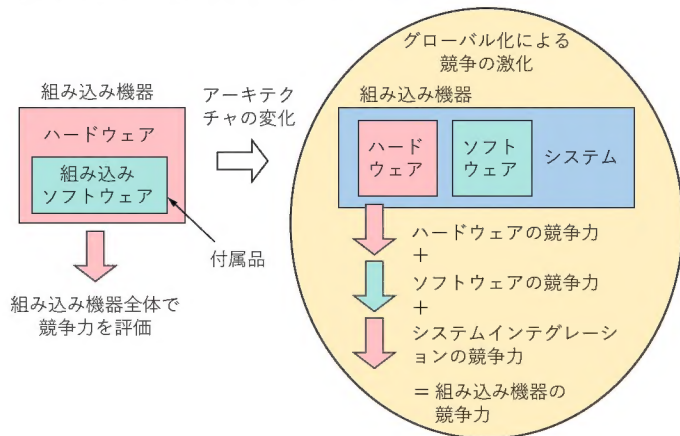
● **人的な取り組みポイント**

組み込みソフトウェア事業を成功させるために，必要な人の最適配置と，その考え方に基づくソフトウェア開発プロセス，さらに，モデル作成者の適性をどのように測るかについて述べる．

2 この特集の背景

● **組み込みソフトウェアの課題が整理されていない！**

この特集を執筆しているメンバー（グループ内では，「組み込み探検隊」とか「組み込み開拓団」などと呼んでいる）が集まってよく議論する話題は，組み込みソフトウェアで現実に何が課題なのかが明確に認識されていないということだった．開発現場は，日々の開発に追われて問題を整理する時間もなく，また自分の担当する製品以外の情報を入手しにくい．コンサルタントは，ミクロなテクノロジに目が向きがちで，事業としての視点に欠ける場合も多い．また，個別のクライアントの問題解決を優先して，全体を見ることが難しい．

〔図2〕製品アーキテクチャの変化と組み込み機器の競争力

● **これでは共倒れ**

このような状態では，個別の企業やコンサルタントがばらばらに問題解決することになり，効率的とはいえない．企業の壁はあるにしても，できるだけ個別の負担を減らし，効率的なアプローチで問題解決にあたりたい．そのほうがお互いにコストも時間も節約しつつ，全体として組み込みソフトウェア技術の底上げにつながるのではないか．このような議論の末，今回の特集が執筆されることになった．

3 なぜ，事業としての組み込みソフトウェアを考えるべきなのか？

● **最初はハードウェアの付属品だった**

1980年代はじめ，つまり組み込みソフトウェアが本格的に製品に使われ始めた時代，ハードウェアの付属品的な扱いをされていた時期がある．

● **この時期は，日本の製造業が世界的に有利な時期だった**

また，この時期はいくつかの要因が重なり，日本の製造業が世界的に有利な立場にあり，日本製の組み込み機器が世界を席巻した[1]．

● **組み込み機器全体をトータルに考えて競争力を議論していた**

大局的に見ればライバル不在だったこともあり，製品としての競争力を議論していれば競争に勝つことができた．とくにソフトウェアは，あくまでもハードウェアを補完する付属品であり，その競争力については，ほとんど議論されてこなかった．

● **製品アーキテクチャの変化**

ネットワーク接続に起因する組み込み機器の高機能化，CPU処理能力の向上などから，組み込みソフトウェアが大規模化し，製品アーキテクチャが変化した(**図2**)．これまでは，ハードウェアの付属品にすぎなかったソフトウェアが，システムを支える重要な要素に変化した．この製品アーキテクチャの変化によって，製品全体の枠組みで競争力を図ることが難しくなった．ハードウェア，ソフトウェア，システムインテグレーションなど，個別の技術ドメインごとに競争力を把握し，弱いところを改善しなければ，競争に勝てなくなってきた．

● **グローバル化と水平分散化**

グローバル化が急速に進展したことにより，多くのライバルが出現した．また，グローバル化にともなって，産業構造が垂直統合型から水平分散型へ移行しつつある．このため，これまでの組み込み機器全体での競争から，ハードウェア，ソフトウェア，部品，システムインテグレーションなどの事業レイアごとの競争に構造が変化してきている．**図3**に，組み込み機器の場合の垂直統合型と水平分散型の産業構造例を示す[2]．

〔図3〕組み込み機器の産業構造：垂直統合型と水平分散型

垂直統合型
ソリューションプロバイダ
アプリケーションソフトウェア
OS/インフラソフトウェア
システムマニファクチャ
部品，システムアーキテクチャ
水平分散型

〔図4〕携帯電話の製品アーキテクチャ

また，携帯電話における水平分散化の例(あくまでも例であり，実際の製品とは異なる)を**図4**に示す．

● **組み込みソフトウェアも独立した事業として考えるべき**

各事業レイア内での競争が激しさを増すことを念頭に置けば，組み込みソフトウェア開発も一つの独立した事業(あるいは，いくつかの独立した事業)として考え，その付加価値を真剣に検討する時期に来ている．

＊　　＊

本特集における開発現場からの報告でも，事業性がしっかり検討されており，開発現場でもこのような認識が共有されていると感じている．今後，どのようにして事業としての組み込みソフトウェアを可視化し，課題を認識し，解決するかを考えなければならない．

参考文献

1) 松田久一，「情報家電産業の再生とリバイバル戦略」，JMR生活総合研究所 提言論文 http://www.jmrlsi.co.jp/index.html
2) 垂直統合型と水平分散型のビジネス・モデル
http://www.atmarkit.co.jp/fpc/special/servertrend/servertrend02.html

いのうえ・たつき　(株)豆蔵
かわぐち・あきら　(株)ガイア・システム・ソリューション
さとう・けいた　(株)デンソー
すぎうら・ひでき　富士ゼロックス(株)
はしもと・たかなり　ソニー(株)

第1章

オブジェクト指向の導入をどう考えたらよいか

組み込みソフトの開発を考えたとき不足していること

杉浦英樹

本章では，組み込みシステムの開発現場にオブジェクト指向を導入する際に，ポイントとなる図や記述の考え方を最初に示し，本章以降の第2章～第5章において，オブジェクト指向──UMLを使った開発の方法について現場技術者の立場から解説する．その際に使われるユースケースモデル，クラス図，シーケンス図などの位置付けを，本章で明らかにする．

（編集部）

1 UMLのモデルが右脳と左脳のバランスで成り立つということ

コミュニケーションを促進することにより，人間の活動が活性化され，作業の効率が改善されることについて，異を唱える人はいないと思う．UML（Unified Modeling Language）は共通言語だから，当然コミュニケーションツールというカテゴリに属するわけである．このモデルの特徴として，直感的に右脳で把握するための図（クラス図：後述）と，論理的にその図の示す内容を確認または検証するための記述（シーケンス図，コラボレーション図：後述）を，必ず対にして構成しているということを理解しておく必要がある．

UMLをコミュニケーションツールとしてとらえた場合，注意すべきは，間違った描き方があり得るということである．相手に何かを伝えるという行為の結果として，作者の意図が読者に伝わらなければ，描き方が間違っているということがいえるわけである．

設計の意図や戦略がモデルから読み取れないようでは，モデル化する意味があるのかないのかという議論に終始してしまい，過去の開発パラダイムとあまり変わらないことになりかねない．ところが，現実に商品開発の現場でモデルの確認を行うと，ほとんどのモデルからは設計意図や戦略，考え方，工夫というものを読みとれない．これは，従来のパラダイムの開発をモデル表現して，可視化しただけにすぎないということを意味している（**図1**）．

左脳で考える論理部分は，モデルの検証（たとえば，実装して本当に動くか否かの検証）ができればよいので，確認手段が確立しやすく，ほぼ問題ない．しかし，右脳で考えるモデリングの部分については，“センスや才能（もって生まれた能力）”という言葉によって片付けられ，オブジェクト指向技術の工業化に向けての課題について，未解決のままであるにもかかわらず，あたかも解決したかのごとく扱われ，その構築方法や思考プロセスの解説は皆無（?!）に等しく，解説するものがあったとしても実務には耐えられないことが多い．

モデリングは本来，現実をそのまま描写すべきで，たとえ設計や実装を考える場面にあっても，自然にとらえた描写モデルを最初から崩すか，または意図的に崩すように作るべきではない．なぜなら，崩したことでその意図を推論する必要が起こり，また推論の結果として，同じモデルに対していくとおりもの解釈が可能になってしまうからである．これでは，作成者の意図を正確に伝えることにはならない．

たとえば，あるデバイス（モータなど）とそれを制御する動作仕様（スペック）の関係は，デバイスを代表するクラスが定義されるなら，仕様はそのクラス自体に定義するもので，動作仕様という概念オブジェクトをわざわざ作って，デバイス≪has a≫

Column 1
もはや課題などといっていられない

一般に，現時点では顕在化していないものの将来起こりそうな問題を「課題」として認識するが，明らかに起こることが予測できれば，それは「問題」としてとらえるべきだろう．とくに，その解決策を考えたとき，パラダイムの変更など，時間のかかる変革をしなければ解決できないことが多い．昨今のTTM（Time To Market）要求を考えると，顕在化している問題と同様の位置付けで，すぐに解決に向けた行動を起こす必要がある．

もちろん，問題には優先順位をつけ，優先順位にしたがった対処をすべきだが，とくに課題としてカテゴライズしてしまうと，近い将来にとりかえしがつかなくなるほど状況が悪化する可能性を否定できないのではないだろうか？高い問題意識と，ためらいなく解決に向けて行動できるバイタリティが求められている．

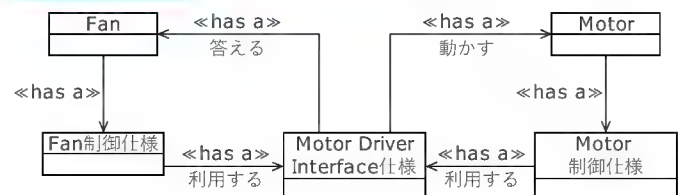
〔図1〕コミュニケーションを促進するためのUML/正しい使い方，間違った使い方(例)

(a) コミュニケーションを促進するためのUML

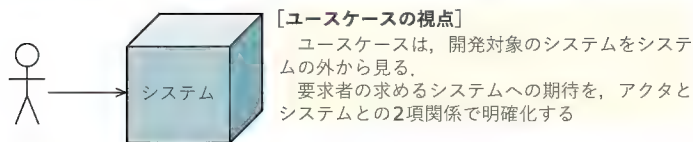
(b) 間違ったモデリングはコミュニケーションを悪化させる

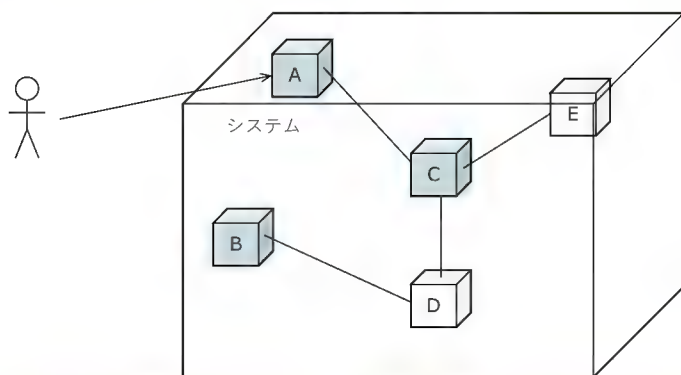
動作仕様というようなとらえ方をすべきではない．これは，現実を描写することによる普遍性の表現による理解の促進を阻害する要因になる．

ドメイン(domain：領域)の経験者であれば，すでにこの事実は理解できているはずだが，オブジェクト指向の最大の特徴である**継承**すなわち≪is a≫の関係について理解できないと，この≪has a≫が示すものが正しく見えてしまうことがあり，注意が必要である．すなわち，再利用によるソフト開発の生産性改善を行う際に，難しい継承によるホットスポットの特定よりも関連による置き換えを行うことが簡単にみえるからである．関連によるアプローチは，オブジェクト指向以前のパラダイムによる開発と大きく変わらないため，下流まで考えた開発効率を求めると，継承に劣ることはいうまでもない．

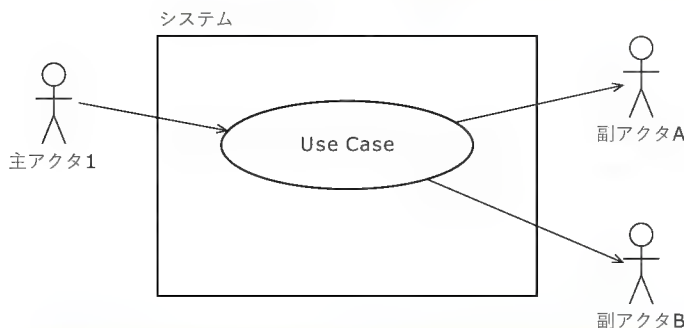
また，親クラスと親クラスの間に関係を作る際，≪Interface≫クラスを安易に作成し，クラス間のオーバヘッドを増すことにどのような意味があるだろうか(**図2**)？　もちろん，インターフェース仕様に着目し，それを抽出する，あるいは十分に分析する必要があるのなら，クラスとしてインターフェースを代表させることに意味があるだろう．そのように考える必要のあるインターフェースがどのくらいあるのだろうか？　われわれはやみくもに本来なら簡単にできるシステムを複雑化しているだけで

〔図2〕Interfaceクラスによるパラドックス：簡単なはずのシステムが複雑化する

〔図3〕ユースケースの考え方，View(視点)と検査への応用

［ユースケースの視点］
ユースケースは，開発対象のシステムをシステムの外から見る．
要求者の求めるシステムへの期待を，アクタとシステムとの2項関係で明確化する

［検査への応用］
システム使用者の立場でのシステムに対する期待のすべてが書かれていれば，システムテストでのテストスクリプトはすべてユースケースシナリオに記述されていることになる．
ただし，現実には実現手段としてのハードウェアの特性を考慮したテストスクリプトが期待されるので，分析ユースケースのシナリオにハードウェア特性による検査項目の追加が必要である．
このことは，システムをハードウェアとソフトウェアが協調して成立させているということからも明らかである．また，システムの品質を検証するのが検査であると考えれば，たとえばタイミングクリティカルなハードウェアの特性を考慮したテストスクリプトを検討するのは当然のことといえる

(a) Step-1：システムの分析

［ユースケースの視点］
システムを構成する複数の部分(モジュール：A～E)の責務に着目し，関連する部分(モジュール)をアクタに見立てて，2項関係でシナリオを構成する

［検査への応用］
システムの結合テスト時には，設計ユースケースに書かれているシナリオのとおりの2項関係が成立していることを確認すればよい．
結合テストのスクリプトとして設計ユースケースシナリオをそのまま利用することができる

(b) Step-2：システムの設計

システム境界を示すBoxが重要になる．
2項関係とは，この境界の中と外の関係を示す．
システム境界の外のアクタは，主アクタ1と副アクタA，副アクタBの三つのロール(role)で表現されているが，アクタという属性でとらえれば一つとなる．
つまり，システムはアクタとだけやり取りをしており，主アクタ1と副アクタA，副アクタBはアクタのロールにすぎないことになる

(c) 2項関係の考え方

はないだろうか？

オブジェクトにより，いままで見えなかったものが見えるようになり，ソフトウェアの制御構造や構成を知ることができる状況になってきたのは，モデリングによる開発手法の貢献にほかならない．しかし，見えるようになったことであたかもエントロピーの法則にしたがうように簡単な制御構造から複雑な制御構造へ変化してしまうことのきっかけになっているとしたら，それは問題である．つまり，よくわからないことが認識されれば，当然，ソフトの構造をシンプルに構成するような管理や努力が払われるが，モデル化されることでソフトの掌握がしやすくなると，より複雑な構造でソフトを作ることに対する拒絶感がなくなることを懸念する．

われわれは，複雑なシステムを簡単にすることを当たり前のこととして評価しなければならない．モーツァルトが作曲するときに，次の音符の絶対性を求めたように，システムを単純化することが普遍を生み出すための近道になることを理解すべきである．とくに組み込み制御機器の開発に求められるのは無駄のない設計，無駄のない動作を基本としたコストパフォーマンスの追及である．この目的を見失うと，手段としてのソフトが目的化し，結果として真の目的を達成するための道筋として遠回りすることにならないだろうか．

最近のヒット商品のめざしたものには“いやし”や“あそび”をテーマにしたものもある．しかし，普遍性を認識しないシステムでいやしを実現しようとしても，何をどうすればいやされるのかがわからないまま時間だけがすぎていくことになるだろう．デカルト思考やブレークスルー思考，システム分析・設計技法などを用いて，必要十分な分析と考察ができた者こそが，開発の結果をヒット商品に結びつけることができるのではないだろうか？

普遍的なモデルを作るための方法について，今回は以降の章で，エレベータの制御モデルと電波時計システムという二つのモデルを例にあげて解説する．とくに，普遍的なモデルと実装モデルの関係を明確にすることにより，制御対象をモデル化するにあたり普遍的な部分と変化する部分との切り分け方法，あるいは概念オブジェクトと実体オブジェクトの切り分け方法を提案するので，UML-オブジェクト指向でクラス図を作成するときの参考にしていただきたい．

本章～第5章で使用するモデルは，ユースケースモデル，クラス図，シーケンス図の3種類である．それ以外にもステートチャート図(状態遷移図)，コラボレーション図などをTPOにあわせて利用する．前者の三つのモデルは重要なので，念のため簡単に解説する．

2 ユースケース

ユースケースの解釈について，ユースケース図とユースケースシナリオを簡単に説明する．

2.1 工程上の位置付け

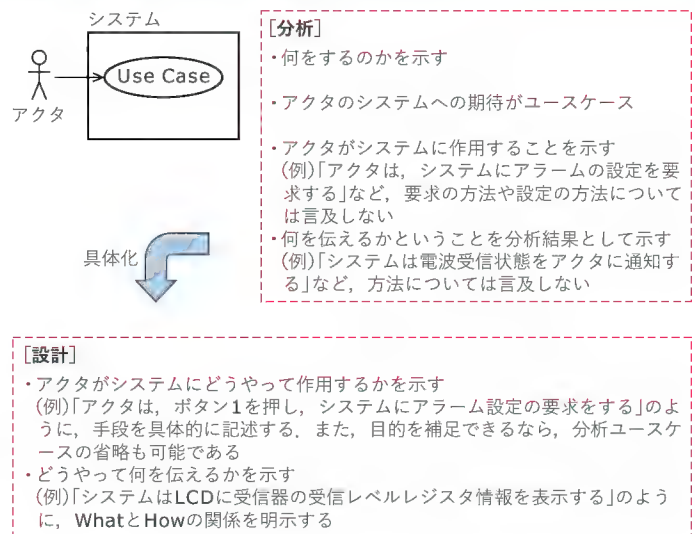
ユースケースは組み込みシステムに対する要求を受け，整理する外部との唯一の接点と位置付ける．現実の組み込みソフト開発においては，ハードウェアシステムのもつ制約条件が複雑かつ具体的であるために，ユースケースでは要求のすべてを表現できないということがよくいわれているが，これはユースケースを利用する側の姿勢の問題である．ユースケースシナリオ自体に必要なデータをすべて埋め込むことも可能だし，詳細な制約条件は別の資料にしてユースケースシナリオからポインタを張ることで要求を整理し，具現化する．これは，過去のパラダイムでも行ってきたことである．

さらにユースケースは，組み込みシステムのふるまいについて具体的に記すことができるので，ソフトとハードを結合した後で，ユースケースシナリオが実現できることを検証できれば，要求どおりに組み込みシステムが動いていることを確認できることになる．つまり，ユースケースシナリオを検査仕様書のベースラインとしてそのまま利用することができるわけである．

2.2 ユースケースの特徴

ユースケースは，システムの利用者とシステムの境界を明確にし，そのシステムで何をしようとしているかについて，トップダウンで分析して詳細化することで，システムに期待されていることを明らかにしていく（**図3**）．詳細化にあたっては，ユースケースごとにシナリオ記述をすることで，それぞれのふるまいの実際をシミュレーションしていく．さらに，基本シミュレーションでは出てこない異常な状況，すなわち来るはずのイベントが来なかった場合，システムが内部イベントを発生できなかった場合などにシステムに期待されているサービスが達成できないことを明らかにし，そのときの処理を明確にする．これは従来のパラダイムでいうところの異常系の仕様を明確にする作業に合致している．

ユースケースは，システムのユーザーの立場からサービス内容を特定するが，ユースケースシナリオはシステム内部で行う内容について明確にするわけであり，自然言語で記述することにより，要求元とのレビューや担当者間でのやり取りを簡単にすることができる．反面，自然言語で記述できるということがあだになり，とくにサービスの階層レベルを意識してユースケースを特定しないと，担当者のスキルによって必要以上に詳しく記述されたり，大まかに記述されたりする．このことが，システム全体の要求分析の作業品質を左右することになる．ユースケースシナリオの粒度のばらつきとして知られるこの問題については，まず，ユースケースの命名法定義，ユースケース図に現れるユースケース名によるその工程での粒度のチェック基準の定義，関連するアクタの抽象度に関する定義について吟味し，それぞれの基準を定義していくことで対処する．

ユースケースは分析，設計とステージを分けて書くほうが，やるべきこと（What）に着目した分析作業と，それを実装する手段（How）に着目した設計作業という二つの作業に目的別に分けられるので，開発中に起こるさまざまな混乱を避けることができる（**図4**）．まず何をするかが明確になることで，従来，実現手段が先行し，目的の模索が開発中に繰り返されることで多発していた手戻りを確実に縮小できる．つまり，仕様確定時期を早めるために，分析から設計という作業の順序を守った開発は効果を発揮する．

〔図4〕分析ユースケースと設計ユースケース

［分析］
・何をするのかを示す
・アクタのシステムへの期待がユースケース
・アクタがシステムに作用することを示す
（例）「アクタは，システムにアラームの設定を要求する」など，要求の方法や設定の方法については言及しない
・何を伝えるかということを分析結果として示す
（例）「システムは電波受信状態をアクタに通知する」など，方法については言及しない

具体化

［設計］
・アクタがシステムにどうやって作用するかを示す
（例）「アクタは，ボタン1を押し，システムにアラーム設定の要求をする」のように，手段を具体的に記述する．また，目的を補足できるなら，分析ユースケースの省略も可能である
・どうやって何を伝えるかを示す
（例）「システムはLCDに受信器の受信レベルレジスタ情報を表示する」のように，WhatとHowの関係を明示する

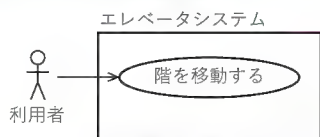
組み込みシステムの開発は，制御対象の要素技術を開発しながら制御ソフトを開発するために，仕様未確定のまま開発作業を進める場面が多く見られる．このようなスタイルで商品開発を行う際にも，新規技術の特定により，確定できないユースケースに着目することで変更箇所を特定できる．開発初期からの変更箇所の特定はソフト開発の上ではホットスポットを特定できることになるので，開発効率の向上に寄与することは間違いない．

2.3 クラス図とユースケース

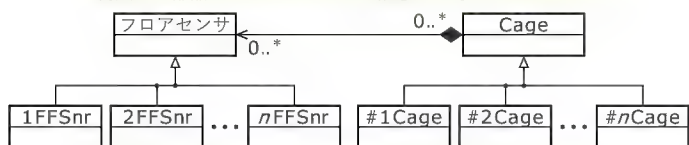
ユースケースとクラス図はビュー（視点）が異なるので，まったく別のものととらえるべきである（**図5**）．組み込みシステムにおいては，機能要件がユースケースビューであり，システム設計上の制約と実現手段を含めたソフトウェアアーキテクチャがクラスビューである．もちろん，分析段階では実現手段の明確化に着目しない場合もあるので，そこで作成されるクラス図は，現実にあるものの関係を明確にする普遍的なモデルということができる．この普遍性の追求が，システムの価値そのものにつながることになる．

2.4 ステートチャート図とユースケース

オブジェクト指向を導入しようとする開発者の中にはステートチャート図をクラスごと作ると考えている方も多いが，これは間違いである．最近話題になりはじめたMDA（Model Driven Architecture）では，クラス図あるいは動的なモデルを作成した後に，コードを自動生成するためのステートチャート図を作成

〔図5〕クラス図とユースケースの視点の違い

分析ユースケースはシステムの利用者の視点で考える．つまり，システム内の実現手段はまったく考慮せずに，アクタとシステムの2項関係でやりたいことを記述する

（a）分析ユースケース

分析クラス図はシステムを開発する側の視点で，システムを構成する部品やサービスといった概念から考える

分析クラス図では制御要素をWhatとして，分析ユースケースの実現手段を考慮していることになる

（b）クラス図

するものが多い．しかも，各ステートにはコードの自動生成を行うための「アクションセマンティクス」と呼ばれる仕様記述言語による記述が必要で，状態にエントリした際の実行内容の記述を行わなければ，コードは自動生成されない．

このアクションセマンティクスの記述方法を統一化することで，自動生成までをサポートするCASEメーカーは，自動生成に関する手続きの統一化をねらっている．しかし，開発ツールの都合で無理に状態として物事をとらえると，本来の分析で得られる普遍性を破壊してしまうことがある．分析を行うための手段としてのステートチャート図が設計や実装の目的になってしまうことに，懸念を感じざるを得ない．われわれは日常，コード生成に用いられるほど厳密な状態認識をしているだろうか？

要求を分析する段階で，状態を考えたほうがシステム仕様を理解しやすいのであれば，ユースケースごとにステートチャート図を用いるべきである．また，たとえばコードを生成するために状態の定義が必要になるなら，それは明らかに設計の作業である．なぜなら，このときに“状態”が必要な理由は，“コードを自動生成する”という目的を達成する手段の制約だからである．

開発者がシステムを理解していれば，間違ったソフトは開発されない．つまり，ソフト開発で発生する欠陥の大半は，ソフト開発者がシステムの動きを理解できていないことにより起こっているのではないだろうか？　システムを理解するために有効な方法であれば，UMLや今まで提案されてきたこと以外のどんな記述方法や方法論を用いてもさしつかえない．なぜなら目的は，要求されるシステムを正しく理解することであり，最終的にはそのシステムを実現し，世の中に製品として提供することだからである（**図6**）．

2.5　アクティビティ図とユースケース

ユースケースシナリオをアクティビティ図で作成するという試みがされているようだが，アクティビティ図はフローチャート＋DFD（Data Flow Diagram）ととらえられるので，作成者はより詳細に記述してしまう傾向がある．一度詳細化されてしまうと抽象化することは難しく，自然言語で記述されたユースケースシナリオに比べると柔軟性に欠ける．また，アクティビティ図による処理の流れはプログラムの流れと直結しやすいので，作成中にクラスの視点による制約が離れないことで，ユースケースとクラス図の独立性を保てなくなるという危険がある（**図7**）．

とくに視点の違いについては注意が必要で，要求のビューに設計のビューが入り込むことで，要求をないがしろにして，設計上の制約を優先するなどといった，旧パラダイムにありがちな間違いがそのまま残ってしまうことになる．このことは本質的に，オブジェクト指向の理解を妨げることにもなりかねない．

さらに，次のステップとしてユースケースの抽象化や拡張，分離を行う場合に，複雑なシナリオの構成であればあるほど論理的には分離しやすいはずなのだが，実務的にはメンテナンスしづらくなることも理解しておきたい．

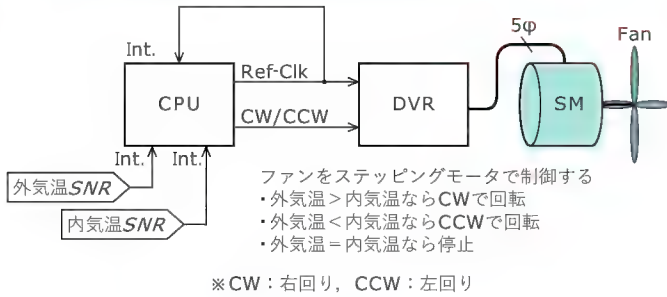
2.6　シーケンス図とユースケース

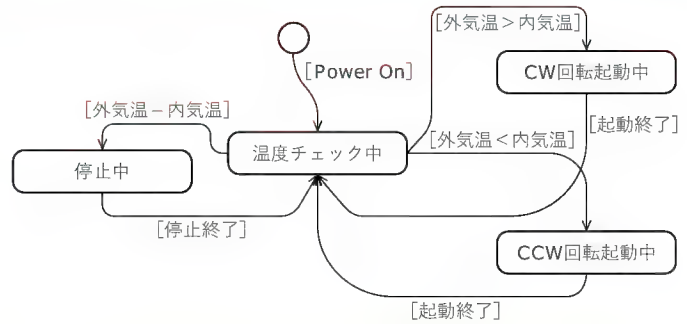
シーケンス図は，クラス図に示されるソフトの構成・構造でユースケースが実現できることをクラス間のメッセージパッシングをトレースすることで確認するための図である．したがって，シーケンス図はユースケースのシナリオの数だけ作成されるべきもので，ユースケースシナリオごとにシーケンス図があれば，間違いなくそのユースケースは開発対象の組み込みシステムで実現できるということが保証できる．

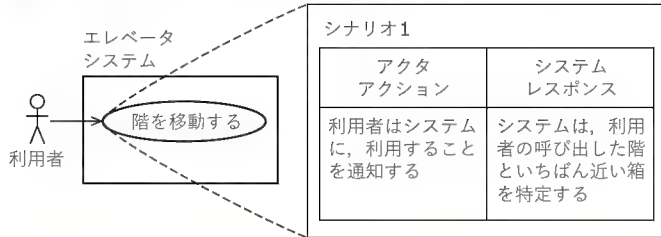
Column 2
不完全な産官学の協業体勢

組み込みソフト開発に関する注目度は上がってきているが，産官学の協業体制の整備は遅れている．欧米や中国に比べ，整っていないと感じている諸兄が多いのではないだろうか？　その原因の一つに，産業界の組み込みソフトエンジニアが，自分たちの開発活動をソフトウェア技術と認識していない部分があるということを感じる．対応する学会としては，情報処理学会ということになるのだが，皆さんは情報処理学会という響きから，自分たちが関係しているということを察することができるだろうか？

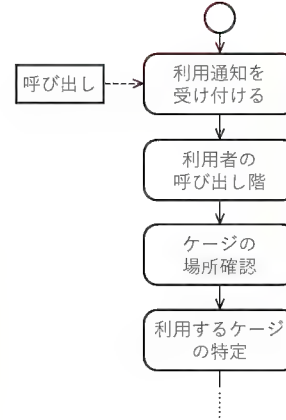
具体的には，情報処理学会（http://www.ipsj.or.jp/）ソフトウェア工学研究会，SESSAME（http://blues.tqm.t.u-tokyo.ac.jp/esw/index.htm），CEST（http://www.ertl.ics.tut.ac.jp/CEST/）などの団体が，組み込みソフト開発に関する技術体系や教育体系の整備に乗り出している．とくに産業界からの情報の入手性が悪いという問題が取りざたされる部分も多いので，興味をもたれた方は積極的に参加すると，それぞれの団体で歓迎されることになるはずである．

〔図 6〕システムを理解するためのモデル

（**a**）制御ブロック図　　（**b**）ステートチャート図

「制御ブロック図」と「ステートチャート図」，どちらがシステムを理解しやすいだろうか？　両方ともシステムを大づかみするためのモデルということができるが，視点が異なる．
ステートチャート図で示すことにより，Motorの起動中と停止中には温度チェックしないほうがシステムがシンプルになるなどの制御上の制約を明確にすることができる

これに加え，クラス図の視点でも確認し，制約について検討できれば，システムの完成度が高まる．
理想的には，さまざまな視点でシステムを見るべきである

〔図 7〕ユースケースシナリオとアルゴリズム，視点の違い，拡張に対する懸念

[**ユースケースの視点**]
・2項関係の明示
・システムの動作の記述はシステムの処理内容を箇条書きで整理する

[**シナリオの拡張性**]
・ユーザーの視点で書くので，要求の変更があれば原点に戻って，シナリオを追加する
・新たな仕様については先入観なしに分析設計できるので発想を広げやすい

（**a**）ユースケース

[**アクティビティ図の視点**]
・フローチャートに近いので，実現手段や手続き方法を考えやすい
・システム内部に着目しがちなので，アクタとの2項関係が明示できない傾向がある

[**シナリオの拡張性**]
・ベースラインが拡張時の思考を，手段や方法に誘導してしまうので，手段や方法の制約が先立ち，その制約の中での拡張性を考えていく傾向がある

（**b**）アクティビティ図

3 クラス図

組み込みシステムに対する要求を噛み砕き，整理した分析ユースケースによって，開発対象となるシステムを把握する．また，設計にあたっての制約条件や，制御するコンポーネントを明確にすることで，コンポーネントとサービスのあるべき関係を示すのが分析クラス図である．

分析クラス図を作成することで，開発対象のシステムが本質的にどのようなモノの集合なのか，また，概念として抽出されるものと，制御対象との関係を明らかにすることで，あるべきソフトの構成や構造を明確にすることができる．

いわゆる“What”系のクラス図が完成したら，そのクラス図を何度も考えることで普遍となる要素を抽出し，キーパターンとしてとらえる．このキーパターンをなるべくそのまま利用し，現実にハードウェアを制御するための制約条件と制御方法に目を向けて作成するのが，設計クラス図である．

設計クラス図はそのまま詳細化して，実装することになるので，実装時の制約まで見通した制約を考慮して作成する．たとえば，ソフトが処理を行うための基本システム（OS）については，ソフト開発者以外から要求が出ることはない．したがって，ソフト開発チームがこの要求を出して，自分たちでその要求を実現することになる．これは，開発全体から見るとソフト上の制約条件として扱われることになる．

OSとアプリケーションの関係をクラス図でどう示すかということに関しては，デザインパターンに関する参考書の大半で扱われるので，必要な方は参考書による調査研究をされることをおすすめする．組み込みシステムでは，“*Real-Time UML, Second Edition Developing Efficient Objects for Embedded Systems*”，和訳版『リアルタイムUML 第2版 オブジェクト指向による組込みシ

〔表1〕CASE Toolメーカーと開発手法

ツール	手法/開発プロセスほか
Rational Rose	OMT，OOSE，RUP，ユースケース駆動
Rational Rose RealTime	ROOM，RUP
I-Logix Rhaphsody	ROPES
Project Technology BridgePoint	Shlaer-Mellor Method

ステム開発入門』，ブルース・ダグラス著，渡辺博之監訳，(株)オージス総研訳 の第5章「アーキテクチャ設計」を参考にすると良い．

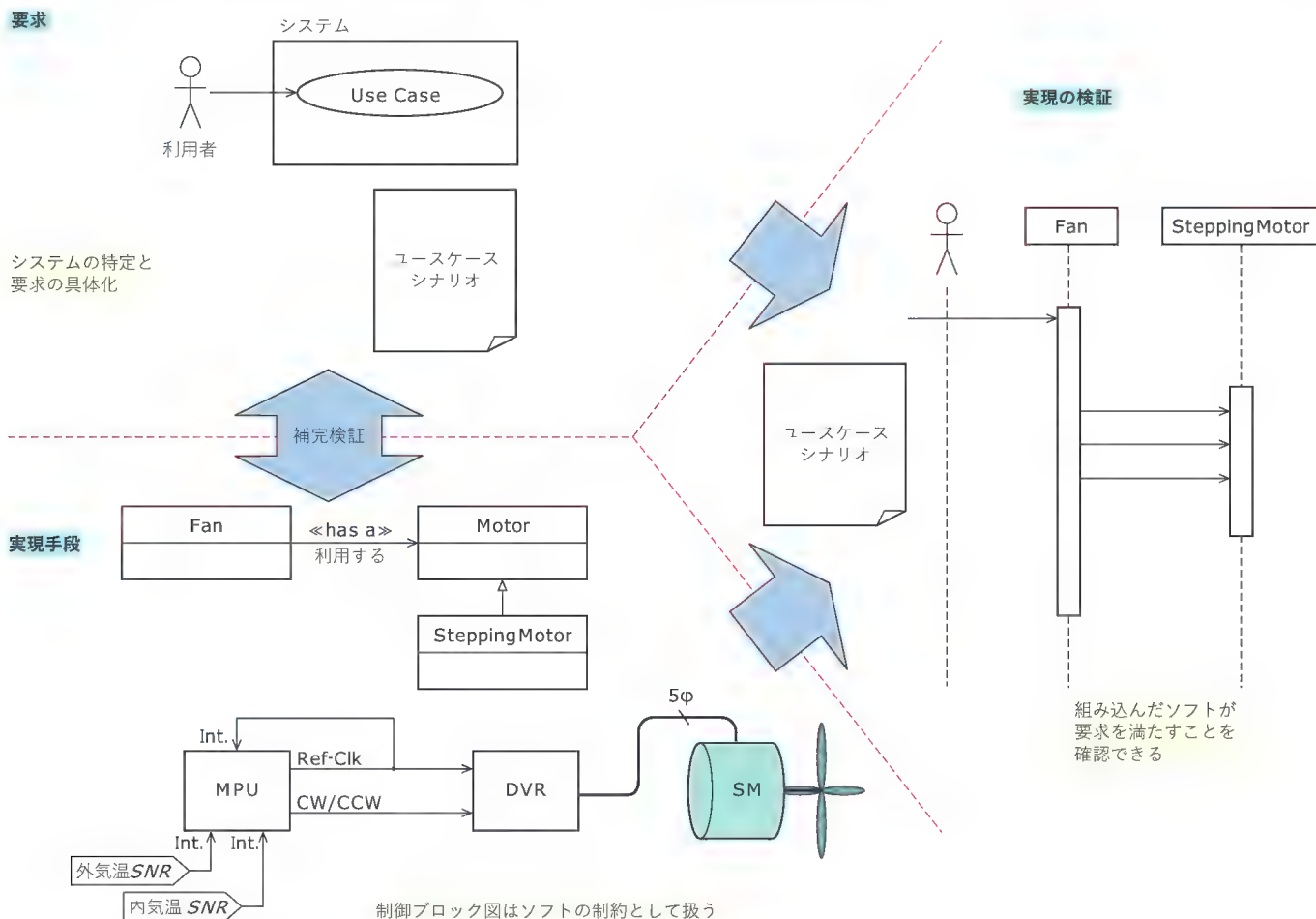
3.1 ソフトを真剣に考える材料としてのクラス図

分析クラス図は，モノとモノとの関係を見きわめるための道具であり，作成してからさらに考えることが肝要である．最初のモデルができあがったからといって満足したのでは，良いモデルにはならない．作成したものを元に，さらに深くクラスの必要性やクラス間の関連を考え，抽象化，具象化を繰り返し行う中で普遍的な形を見出すことができる．このようなプロセスを経ることによって，堅牢でかつ柔軟性に富んだモデルが作成できる．

設計クラス図においても同様なことがいえるわけで，たとえば，C++のソースコードを作成するためにモデルを作成するなどというのは，本末転倒と考えるべきである．機能を実現するための手段を実装する方法が設計クラス図から読み取れなければ，開発者同士で実装方法に関する考えを共有することができず，チームワークで品質を獲得する際の開発効率化には結びつかない．

とくに，前述したMDAのように，メッセージパッシングで動作しないクラス図(メソッドが存在しないで，実装できるもの)では，クラス図の詳細化による実装との一貫性がツールに依存するため，小規模な組み込みシステムをソースコードデバッガで開発するような状況では，開発中のメンテナンス性や仕様変更に対する柔軟性で劣る傾向を否めない．

開発環境の変化を考えた場合にも，利用しているCASEツールのバージョン変更や種類を変える場合にクラス図がそのまま利用できなくなり，何のためにモデル化したのかわからなくなってしまう．短期的な視点でソースコードエディット作業の時間を節約したいからといって，短絡的にコード自動生成ツールを利用した開発を選択すると，このような原理原則部分の間違

〔図8〕ユースケース，クラス図，シーケンス図による検証

いに気付くのが遅くなり，知らないうちにCASEツールメーカーに囲い込まれてしまうというリスクがあることを忘れたくない(**表1**).

3.2　ステートチャート図とクラス図

すべてのクラスが状態により動くと考える必要はないので，クラスごとにステートチャートが必要ということはない．状態で考えるということは，状態を遷移させるためのイベントが必要になってくるが，複数の状態と複数の遷移がある場合に一つのインターフェースメソッドで構成されるクラスの状態はどのように示せるだろうか？

組み込みシステムをオブジェクト指向で実装する場合，理想的にはデバイスを直接制御するクラスとそのクラスを利用するクラスを独立させることで，カプセル化を実現する．したがって，デバイスを直接制御するクラスを利用するクラスが，そのクラスの状態を把握する必要はない．

すべてのクラスが状態で制御されるということは，メッセージによりサービスを提供するといういちばんシンプルな構成を実現する際に，状態の認識と処理の特定という処理のオーバヘッドを生むことになる．

さらに，インクリメンタルに開発を行う(必要な部分のみを段階的に開発する)場合の実装方法については注意が必要で，継承関係にあるクラスでの状態の共有やサブステートの定義による状態定義の複雑化は素直に実装した場合には必要ないのではないだろうか．

3.3　アクティビティ図とクラス図

クラスのメンバであるメソッドにはアルゴリズムがあり，アルゴリズムを明確にするにはアクティビティ図が適しているといえる．アクティビティ図は処理フローとデータフローを同時に記述できるので，クラスのふるまいについてアクティビティ図で表現することも，効率的に開発するための一助になる．

4　動的検証モデル

4.1　シーケンス図

シーケンス図は，クラス図のソフト構成でユースケースシナリオが実現できることを検証するものである．ユースケースシナリオごとにシーケンス図が完成していれば，要求を実装できることの確認が実装前にできているということになる．

レビュー時には，シーケンス図の存在を確認するだけで，時間をかけずにソフト品質をある程度判断することができる．

シーケンス図はクラス図，ユースケースと連携し，三つのモデルが開発対象を成り立たせているということを忘れてはいけない(**図8**)．

すぎうら・ひでき　富士ゼロックス(株)

New Products ― 富士通製プロセッサFR-V用RTLinux/Pro
(株)エフエスエムラボ・ジャパンと(株)アックスは，富士通製プロセッサ「FR-Vファミリ」用の「RTLinux/Pro for FR-V」を日本国内向けに発売した．RTLinux/Pro for FR-Vは，FSMLabs社によって開発され，LinuxカーネルにはアックスがR開発したaxLinux/Pro for FR-Vを使用している．

第2章

機能仕様書をユースケースシナリオと位置付けて分析・実装する

エレベータモデルの検証

本章では，機能仕様書として作成するものをユースケースシナリオとして位置付ける考え方で，具体的な開発例として，エレベータモデルのユースケースを作成する．3階建て/5階建ての建物の2基のエレベータを想定している．クラスの実装例も紹介し，詳しく解説する(全ソースコードは付属CD-ROM InterGiga No.30に収録)．

(編集部)

1 ユースケースでどこまで表現できるのか？

ユースケースとは，システムがアクタに提供する機能のことであると考える．エレベータ制御システムにおけるユースケースの作成にあたっては，実際にエレベータを利用するときの場面を思い浮かべながらユースケースを抽出する．エレベータ制御システムに対して行う主アクタのアクションは，次の四つである．

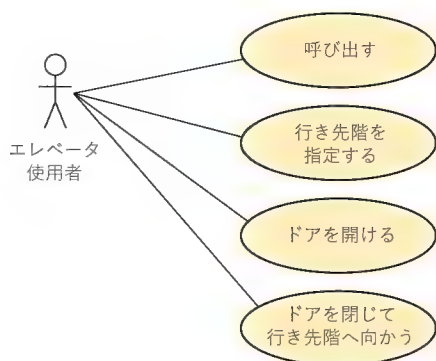
- 呼び出しボタンを押す
- 行き先階のボタンを押す
- ドア閉ボタンを押す
- ドア開ボタンを押す

上記四つのアクションに対するエレベータ制御システムのレスポンスを，ユースケースとして抽出する(**図1**)．

ユースケース名はそれぞれ，次のようにした．

- 呼び出す
- 行き先階を指示する
- ドアを閉じて行き先階へ向かう
- ドアを開く

それぞれのユースケースに対し，シナリオを記述した(**図2**～**図4**．ユースケース「ドアを開く」は省略した)．

〔図1〕
ユースケース図

しかし，前述した四つのユースケースは，エレベータ制御システムがアクタに提供する機能を羅列しただけで，システムを開発するうえで肝心な「仕様」の部分がない．「システムがアクタに提供する機能の仕様」，つまり一般に機能仕様書として作成するものが，ユースケースシナリオとして位置付けられるのではないだろうか？

このような考え方でユースケースおよびユースケースシナリオを作成すると，ユースケースおよびユースケースシナリオのみでシステムがアクタに提供する機能仕様をすべて表現することができる．本章で例としてあげるエレベータ制御システムは，上記の考え方でユースケースおよびユースケースシナリオを作成した．今回は，次に挙げるものをシステム仕様として各モデルを作成した．

- 呼び出しボタンは各階に上下一つずつ．ただし，最上階には下向きのみ，最下階には上向きのみ設置する
- 到着を知らせるランプは，各階ホールのエレベータごとに上向き一つ，下向き一つ．ただし，最上階には下向き，最下階には上向きのみ設置する
- 行き先階指示のためのボタンは，各エレベータ箱内に止まることのできる階数分設置する
- 扉開ボタンは，各エレベータ箱内に一つずつ設置する
- 扉閉ボタンは，各エレベータ箱内に一つずつ設置する
- 現在階表示ランプは，各エレベータ箱内に止まることのできる階数分設置する
- エレベータの進行方向を示すランプは，各エレベータ箱内に上向き一つ下向き一つを設置する
- 各エレベータに扉は一つ．各階の各エレベータの外扉は，内扉とメカ的に連動するしかけとする

2 エレベータモデルの作成

以降，エレベータ制御システムのモデルを作成するにあたっ

〔図2〕呼び出すシナリオ

Use Case Document format.1

エレベータモデル	版No.	発行日	発行者	承認者
	K0		野原 有人	

ユースケース：呼び出す

○システム
・エレベータモデル
○アクタ
・エレベータ-使用者(主アクタ)
○目的
・利用するためにエレベータを呼び出す
○概要
・呼ばれた階に適当なエレベータを移動する
○事前条件
・なし
○事後条件
・ドアを開け，到着階にある到着を知らせるランプを点滅させ，到着階にある呼び出しボタンのランプを消す
○シナリオ記述
シナリオ1：呼び出された階に，停止しているエレベータが存在する場合
▶基本系列

アクタアクション	システムレスポンス
1．アクタは，エレベータホールの呼び出しボタン(上向きor下向き)を押す	2．システムは，呼び出された階に停止しているエレベータがあれば，ドアを開け，到着階の到着を知らせるランプを点滅させ，到着階の呼び出しボタンのランプを消す

シナリオ2：呼び出された階に，停止しているエレベータが存在しない場合
▶基本系列

アクタアクション	システムレスポンス
1．アクタは，エレベータホールの呼び出しボタン(上向きor下向き)を押す	2．システムは，呼び出されたボタンと進行方向が同じで，かつ呼び出された階に止まる予定のあるエレベータがあるか検索する．あれば，シナリオ終了
	3．ない場合，現時点で停止しているエレベータの中から，いちばん近い階にあるエレベータを向かわせる
	4．停止しているエレベータがない場合，最初に停止したエレベータを向かわせる
	5．呼び出された階にエレベータが到着すると，ドアを開け，到着階の到着を知らせるランプを点滅させ，到着階の呼び出しボタンのランプを消す

て行ってきたことを，分析から実装まで順に示す．

2.1 部品仕様と製品仕様

● 部品仕様

エレベータ制御システム(以下システムと呼ぶ)を制御するソフトウェアの作成にあたって，ソフトウェアにとっての制御対象となる部品とシステムの仕様の概要を決めておく．まず，制御対象となる部品仕様の概要を次に記す．

① タイマパルス装置(TPU ： Timer Pulse Unit)

入力クロックをカウントするカウンタである．カウント値を設定可能．設定されたカウント値をカウントし終えると，トグル波形を出力する．また，カウント終了を知らせる信号も出力可能．

② DMA装置(DMAC ： Direct Memory Access Controller)

転送元のアドレスから転送先のアドレスへ指定された転送長分データを転送する．外部からの入力信号を1データ転送のための起動要因とする．本システムでは，転送起動要因に，CPUのポートを介したTPUのカウント終了信号を使用する．

③ センサ

外部からの刺激を検知して，信号出力のレベルを変化させる．

〔図3〕行き先階を指定するシナリオ

Use Case Document format.1

エレベータモデル	版No.	発行日	発行者	承認者
	K0		野原 有人	

ユースケース：行き先階を指定する

○システム
・エレベータモデル
○アクタ
・エレベータ-使用者(主アクタ)
○目的
・行き先階を指示する
○概要
・指示された階を止まる予定階とする
○事前条件
・なし
○事後条件
・行き先階のボタンのランプが点灯している
○シナリオ記述
シナリオ1：
▶基本系列

アクタアクション	システムレスポンス
1．アクタは，エレベータに乗り，行き先階のボタンを押す	2．システムは，行き先階のボタンを点灯する

〔図4〕ドアを閉じて行き先階へ向かうシナリオ

Use Case Document format.1

エレベータモデル	版No.	発行日	発行者	承認者
	K0		野原 有人	

ユースケース：ドアを閉じて行き先階へ向かう

○システム
・エレベータモデル
○アクタ
・エレベータ-使用者(主アクタ)
○目的
・ドアを閉じて，行き先階へ向かう
○概要
・ドアを閉じて，移動を開始する
○事前条件
・ドアが閉じていない
○事後条件
・自動的にドアを閉じるための制限時間のカウントを開始する
○シナリオ記述
シナリオ1：
▶基本系列

アクタアクション	システムレスポンス
1．アクタは，エレベータ内の閉じるボタンを押す．あるいは，制限時間が過ぎて自動的に閉じる	2．システムは，次に向かう階があるか検索する，なければ，シナリオ終了
	3．ある場合，エレベータをその階に移動する
	4．システムは，向かった階にエレベータが到着すると，ドアを開け，到着した階の到着を知らせるランプを点滅させ，到着階の呼び出しボタンのランプを消し，エレベータ内の到着階の行き先ボタンのランプを消す
	5．システムは，自動的にドアを閉じるための制限時間のカウントを開始する

本システムでは，出力信号をCPUの割り込みポートに接続する．

④ **A社製モータ**

所定の周波数のパルス信号を入力されると，入力した周波数に応じた回転数で回転するステッピングモータである．加速は速度の増加にしたがって徐々に周波数を上げていき，等速は一定の周波数を入力する．減速は加速の逆を行うことで回転数を制御する．

⑤ **CPU**

上記①～④に説明した外部接続機器は，CPUのポートを介して接続する．ポートの入出力で外部接続機器を制御する．

● **製品仕様**

次に，製品仕様を示す．

① **到着通知ランプ**

各階に，エレベータ箱ごとに上向き下向き一つずつ設置する．ただし最上階には下向きのみ，最下階には上向きのみを設置する．呼び出しボタンが押されたとき，迎えに来るエレベータがどれかを知らせるために点灯する．そのエレベータ箱が到着したときに点滅し，到着したエレベータのドアが閉まると消灯する．

② **呼び出しボタン & ランプ**

各階に上向き下向き一つずつ設置する．ただし最上階には下向きのみ，最下階には上向きのみ設置する．ボタンを押すとボタン内のランプが点灯する．呼び出した方向のエレベータが到着するとランプを消灯する．

③ **行き先階指示ボタン&ランプ**

各エレベータの箱内に階数分設置する．ボタンが押されるとボタン内のランプが点灯する．ボタンが押された階に到着するとランプを消灯する．また，最上階，最下階に到着するとすべてのランプを消灯する．

④ **現在階表示ランプ**

各エレベータの箱内に階数分設置する．現在階あるいは通過階の階数ランプが点灯する．

⑤ **扉開閉ボタン&ランプ**

エレベータ箱内に開，閉一つずつ設置する．エレベータ停止中にのみ，押されると押されたボタンのランプを点灯し，扉の開け閉めを行う．ボタンを離すとランプは消灯する．

〔表1〕回転/秒等速回転時の加速データ

加速段数	周波数（Hz）	周波数を出力するためのCPUクロックのカウント数	パルス数
1	100	100	1
2	200	50	2
3	300	33	4
4	400	25	6
5	500	20	9
6	600	17	12
7	700	14	16
8	800	13	20
9	900	11	25
10	1000	10	*

⑥ **内扉**

各エレベータ箱に対して一つずつ設置する．A社製モータにて駆動する．

⑦ **外扉**

各階にエレベータ箱ごとに一つずつ設置する．ただし，内扉にメカ的に連動して動くため，制御対象とはならない．

⑧ **エレベータ箱**

人の乗る箱のこと．A社製モータにて駆動する．

2.2 A社製モータ制御方法

A社製モータの制御方法を説明するために例を挙げる．たとえば10回転/秒の等速回転をさせるためには，スローアップ制御を行う．加速をスムースに行わせるために周波数を徐々に高くしてモータ駆動回路に入力する基準パルス信号と，10回転/秒の等速回転に必要な周波数のパルス信号の入力が必要である．

CPUのTPUのトグル出力モードでスローアップ制御を行う際に必要な設定データをリスト化したものを**表1**に示す．ただし，10回転/秒の等速回転時の周波数を1kHz，CPUのクロックを10kHzとする．

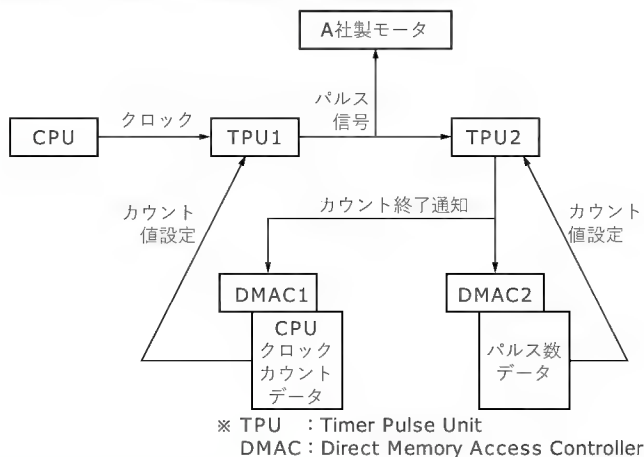
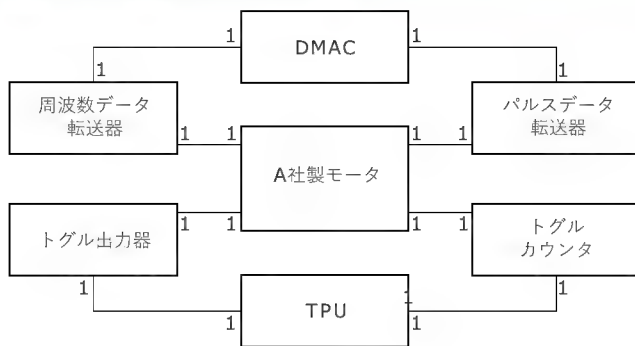
表1の「加速段数」に示された順に「周波数」の欄のパルス信号を「パルス数」分入力することで，A社製モータはスムースに加速を行う．実際に制御に使用するデータは「周波数を出力するためのCPUクロックのカウント数」と「パルス数」である．**表1**に記したデータでTPUのトグル出力を用いて，A社製モータを駆動するための装置の構成を**図5**に示す．

まず，**表1**の「周波数を出力するためのCPUクロックのカウント数データ」の加速段2をDMAC1の転送元とし，転送先をTPU1のカウント値設定部とする．TPU1のカウント設定部には加速段1の「周波数を出力するためのCPUクロックのカウント数データ」をセットしておく．

同様に，**表1**の「パルス数」の加速段2をDMAC 2の転送元とし，転送先をTPU2のカウント値設定部とする．TPU2のカウント設定部には加速段1の「パルス数」をセットしておく．TPU1

〔図5〕A社製モータを駆動するための装置の構成

〔図6〕モータ駆動のためのクラス図

〔図7〕TPUおよびDMACの使用者との関係

はCPUからのクロックを入力クロックとし，出力信号はA社製モータとTPU2に入力されている．TPU2はTPU1のトグル出力を入力クロックとしている．

また，DMAC1，DMAC2ともにTPU2のカウント終了を知らせる信号が，データ転送の起動要因になっている．このように動作開始前の設定を行って，TPU1/2，DMAC1/2を起動する．

TPU1はCPUのクロックをカウントして100になったら，100Hzのトグルデータを出力する．TPU1のトグル出力をTPU2はカウントしている．最初のカウント値は1なので，TPU1のアクティブエッジを検出したTPU2はカウント終了信号を出力する．DMAC1はTPU2からのカウント出力終了信号を要因に加速段2の「周波数を出力するためのCPUクロックのカウント数データ」50をTPU1のカウント値設定部に転送する．

同様に，DMAC2はTPU2からのカウント出力終了信号を要因に加速段2の「パルス数」2をTPU2のカウント値設定部に転送する．するとTPU1は，CPUのクロックをカウントして50になったら，200Hzのトグルデータを出力する．TPU2はカウントアップし，TPU1は再度のカウント終了で200Hzのトグルデータを出力する．

TPU2は，このとき「カウント値」2なので，TPU2はカウント終了信号を出力する．DMAC1はTPU2からのカウント出力終了信号を要因に加速段3の「周波数を出力するためのCPUクロックのカウント数データ」33をTPU1のカウント値設定部に転送する．同様に，DMAC2はTPU2からのカウント出力終了信号を要因に加速段3の「パルス数」4をTPU2のカウント値設定部に転送する．

以上のような方法で，TPU1から所定の周波数のトグル信号を出力し，A社製モータを加速することができる．10回転/秒を維持したいパルス数を**表1**の*に設定すれば，その期間10回転/秒を保持する．停止する場合は，加速の逆の制御を行う．

2.3　クラス抽出

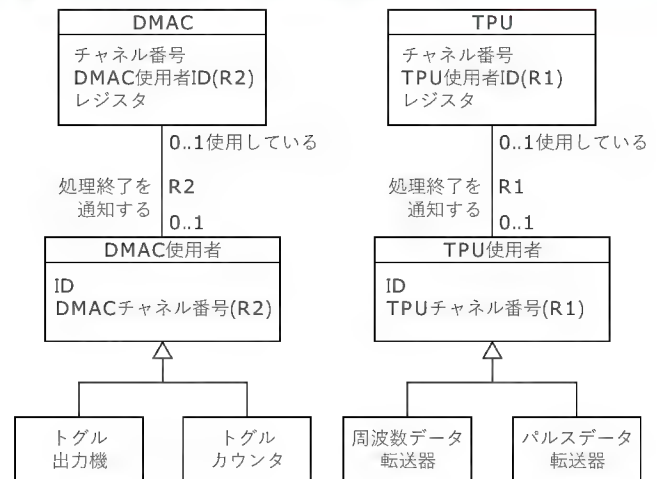
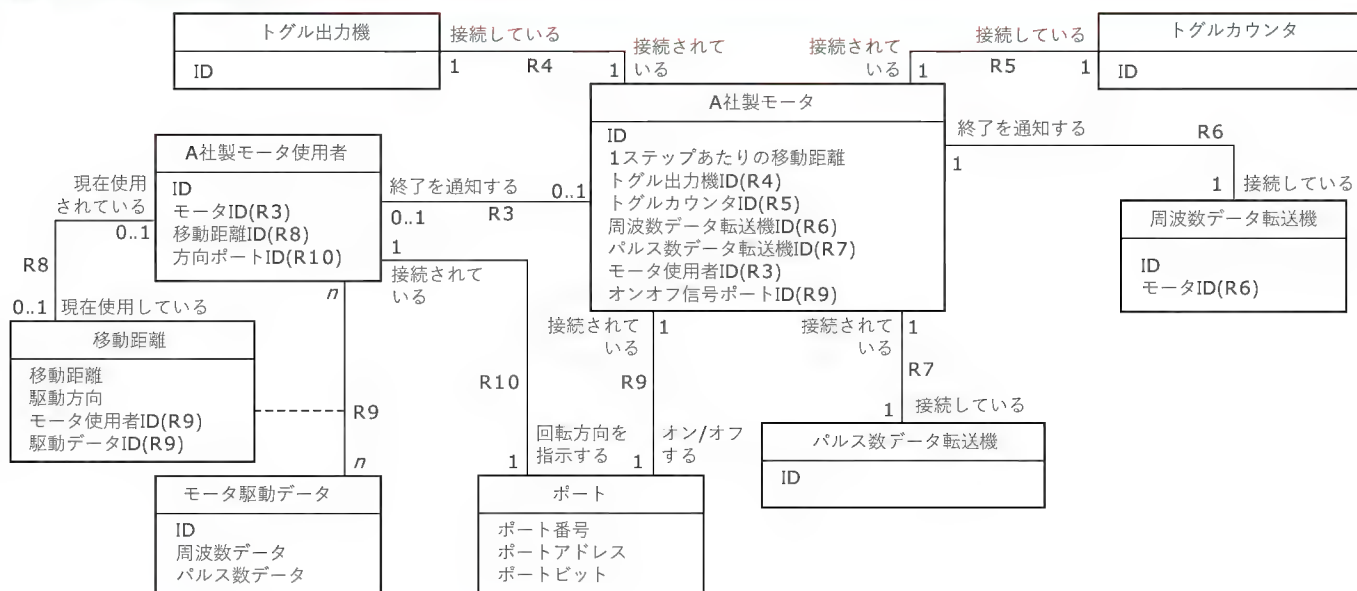
まず，A社製モータ駆動部分のクラスを抽出する．「A社製モータ」そのもの，A社製モータにトグル信号を出力する「トグル出力器」，A社製モータに出力したトグル出力による矩形波のエッジをカウントする「トグルカウンタ」，所定の周波数のトグルデータを出力するためのデータをトグル出力装置に転送する「周波数データ転送器」，同様に，出力するトグル数を転送する「パルスデータ転送器」，「TPU」，「DMAC」を抽出した(**図6**)．

モータ駆動のためのクラス図では，DMACやTPUを使用するクラスが増えた場合にクラスとして抽出した「DMAC」や「TPU」への関係が増えてしまったり，「TPU」と「DMAC」から処理終了を通知したい場合にどのクラスに通知してよいかわからなくなってしまう．そこでこれらのコンポーネントの再利用性・拡張性に着目して整理する．

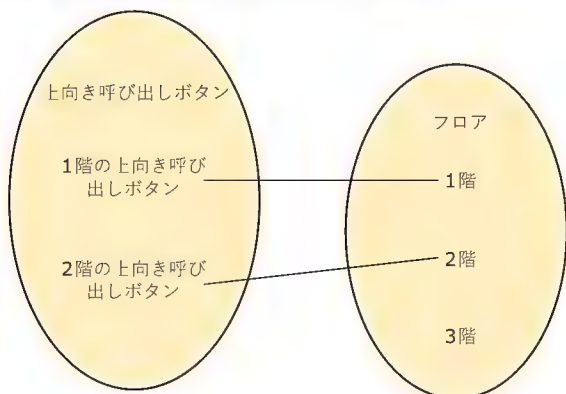
図7のようにすることでDMAC，TPUは使用する側に影響されず，使用者側も自由に増やすことができるように構成できる．その結果を反映させたのが，**図8**である．

A社製モータとA社製モータ使用者の関係はDMACとDMAC使用者の関係と同様である．A社製モータを使用するのはエレベータ箱の移動と扉の開閉である．**図8**に示すR3の関係を用い，後に出てくる扉クラスとエレベータ箱クラスを「A社製モータ使用者」クラスのサブタイプとすることで，とくに問題は起こらないはずである．

「モータ駆動データ」クラスは，モータ回転開始から停止までの周波数データとパルス数データを所有している．システムにおいて，エレベータの移動する距離は限られた数値データになる．その距離ごとに最適なモータ駆動データを選択できるように「移動距離」クラスを「A社製モータ使用者」クラスと「モータ駆動データ」クラスの関連付けクラスとして抽出した．この「移動距離」クラスの関係によって，A社製モータ使用者は移動距離と移動方向を決定するので最適なモータ駆動データを選択することができる．

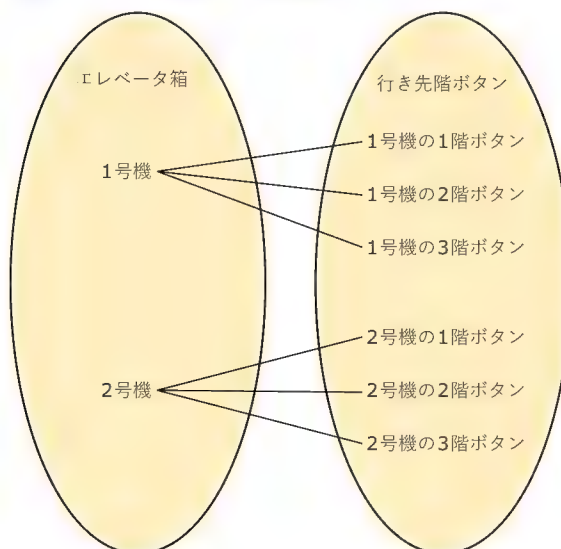
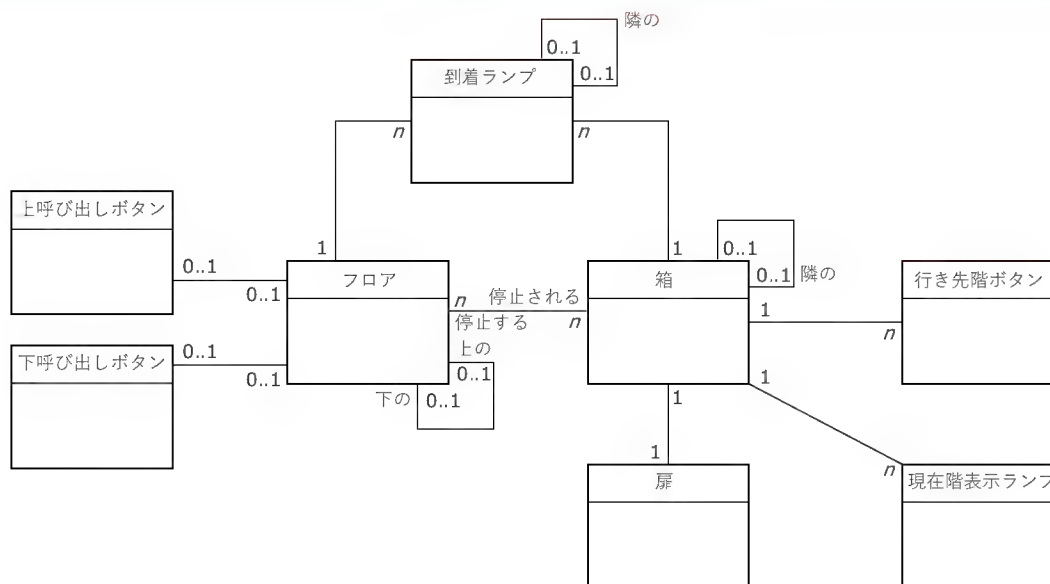
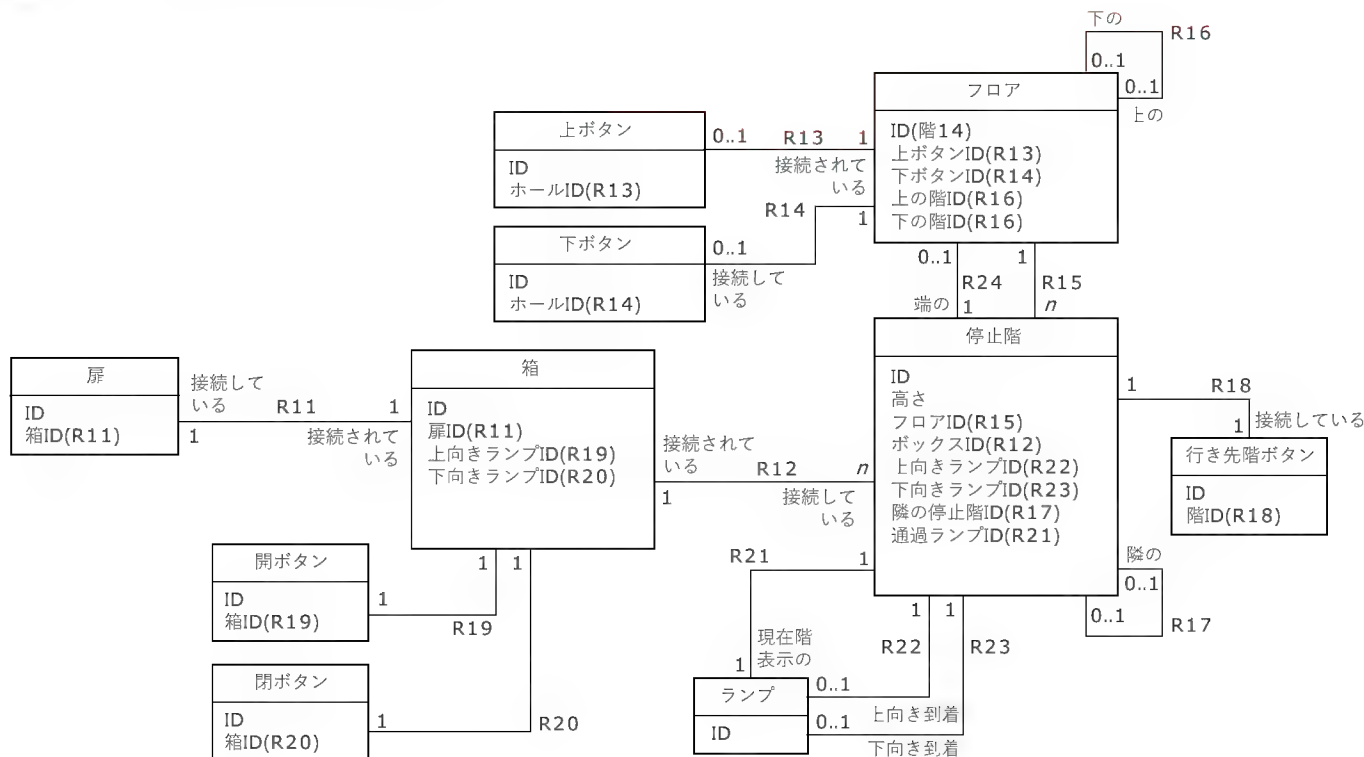
次にエレベータ制御側のクラスを抽出する．製品仕様から，順次制御に必要なクラスを抽出してみる．「上向き到着ランプ」，「下向き到着ランプ」，「上向き呼び出しボタンとランプ」，「下向き呼び出しボタンとランプ」，「行き先階指示ボタンとランプ」，

〔図8〕A社製モータ駆動モデル

〔図9〕上向き呼び出しボタンとフロアの関係

〔図10〕5階建ての場合の上向き呼び出しボタンとフロアの関係

「扉開ボタンとランプ」，「扉」，「エレベータ箱」を抽出した．加えて，1階2階などの「フロア」をクラスとして抽出した．

さらに，抽出したクラスのインスタンスについて考察する．例として3階建てのビルに2機のエレベータがある場合を考えた．上向き到着ランプは各階，各エレベータ箱に一つずつで，最上階はないので，合計4個のインスタンスとなる．

同様に下向き到着ランプも4個，上向き呼び出しランプは最上階を除く各階にあるので二つ，下向き呼び出しランプも二つ，行き先階ボタンは各エレベータ箱に全階分あるので6個，扉開ボタンは各エレベータ箱に1個ずつなので2個，閉ボタンも2個，エレベータ箱は2個，扉も2個，フロアは3個のインスタンスが存在している．たとえば，上向き呼び出しボタンは各フロ

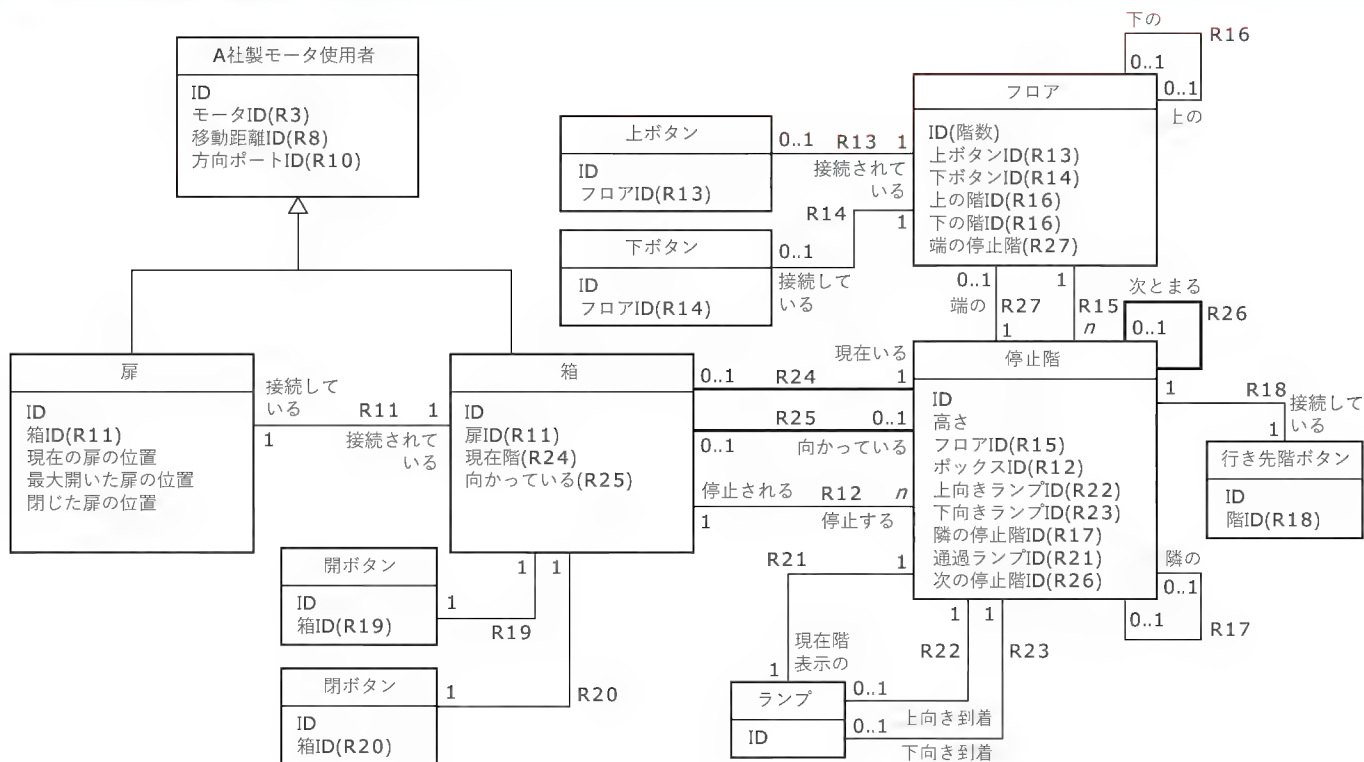
〔図11〕エレベータ箱と行き先階ボタンの関係

〔図12〕エレベータ制御モデルのクラス図(その1)

〔図13〕エレベータ制御モデルのクラス図(その2)

アに一つずつあるので，上向き呼び出しボタンとフロアの関係を考察する(**図9**).

ここでは，3階建てについて考察してみたが，もし5階建てならどうだろう．5階建てについても考察してみた(**図10**).

3階建てと5階建ての例から，上向き呼び出しボタンとフロアの関係は“0..1”対“0..1”とできることがわかる．同様に，下向き呼び出しボタンとフロアの関係を“0..1”対“0..1”とした．

同様に，エレベータ箱と行き先階ボタンについても考察する(**図11**).

エレベータ箱と行き先階の関係を書き出すと1対nの関係になることがわかる．ほかにも，いろいろなエレベータを利用するときの状況を思い浮かべて，物理的に存在する場所に関係がありそうなもの，順序性がありそうなものを関係させて考察してみた．その結果を反映させたクラス図を**図12**に示す．

〔図 14〕エレベータ制御モデルのクラス図(その 3)

クラス図を見ると，箱とフロアと到着ランプの関係に n の関係が多いこと，関係が3者間でグルグル回っていることがわかる．さらに，各フロアの各エレベータ箱が停止する階を「停止階」クラスとして考えた．抽出したクラス・インスタンス間の関連をもう一度検討しなおした結果のクラス図が**図 13**(前頁)である．

「箱」と「停止階」の関係は，「現在停止している」という関係や，移動中であれば，「次に停止する」という関係もある．また，現在移動中の箱が止まる予定の停止階は複数ある．これらの関係を追加したクラス図を**図 14** に示す．

このクラス図の R24 と R15 の関係で，エレベータ箱が現在どのフロアにいるかを一意に特定できる．また，移動中のエレベータ箱であれば，R25 で関係する停止階が存在する．R25 の関係先がないものは，移動中でないことがわかる．複数の停止予定階がある場合は，R26 の関係で調べることができる．

2.4 クラス図の検証

ユースケース「呼び出す」のシナリオを例にクラス図を検証する．「呼び出す」のシナリオを実現するには，どのような情報が必要だろうか？

① 利用者の行きたい向きの情報
② 利用者のいる階数の情報
③ すべてのエレベータ箱の現在いる階の情報
④ すべてのエレベータ箱の状態．どこに向かおうとしているのかの情報
⑤ どこかに向かおうとしているエレベータ箱が次に止まる階の情報
⑥ どこかに向かおうとしているエレベータ箱の止まる予定の階の情報

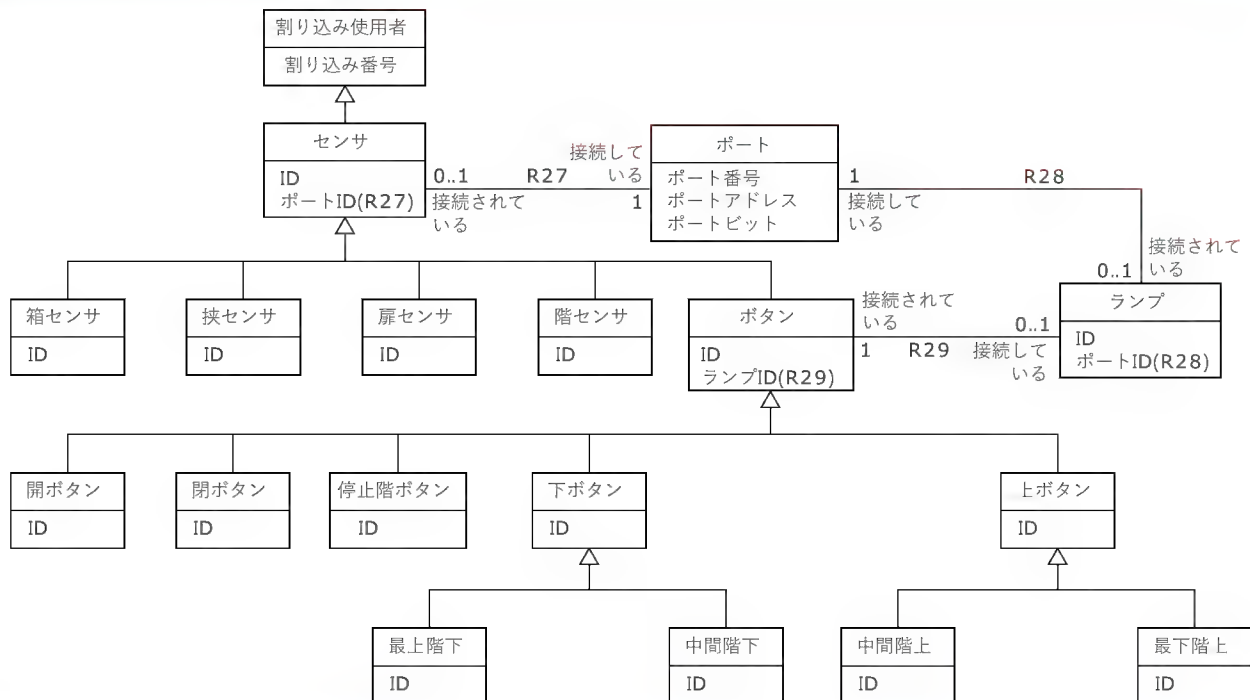
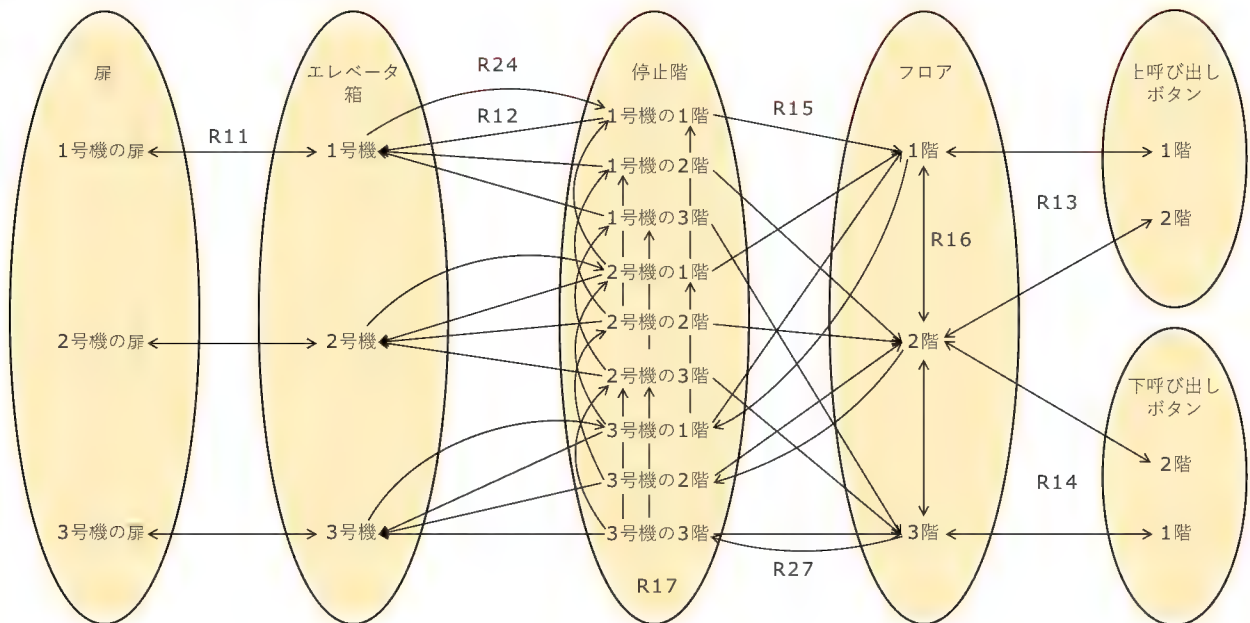
①～⑥の情報が**図 14** からわかるだろうか？

検証の前に，実際にボタンが押された場合に，押されたことを検知するためのモデルを追加する(**図 15**)．

クラス「割り込み使用者」は属性である割り込み番号の外部割り込みが発生すると，割り込みが発生したことを CPU から通知される．今回対象としている CPU では，外部割り込みが発生すると，0x000000 番地からの割り込みベクタに登録されているアドレスにジャンプするとともに，割り込み信号のレベルを CPU ポートに入力する．

実装方法として，クラス「割り込み使用者」の割り込み通知を受ける操作のアドレスを，割り込み番号に対して割り振られているベクタアドレスとして書き込んでおく．たとえば割り込み番号 2 番の割り込みが発生すると，割り込み番号 2 番用のベクタアドレスに書き込まれている番地にジャンプする．その結果，割り込み番号 2 番用のベクタに登録されている操作が処理されることになる．

ソフトウェアは，割り込み通知を受けると，通知を受けた割り込み番号に対応するポートを確認することで立ち上がりエッジか立ち下がりエッジかを検知することができるようになっている．クラス「センサ」は「割り込み使用者の」サブクラスで，割り込み確認のためのクラス「ポート」と関係している．クラス「セ

〔図15〕割り込み検知のためのクラス図

〔図16〕各クラスのインスタンスの関係

ンサ」のサブクラスで，いろいろなセンサを準備しているが，これらは実際にシステムを制御しようとすると必要になるクラスである．

クラス「ボタン」もセンサのサブクラスである．ボタンは通常ランプと一緒になっているのでクラス「ランプ」と関係できるようになっている．クラス「ランプ」はクラス「ポート」と関係をもっており，実際のランプのオン/オフ処理はポート出力で行うようにしている．

2階の上向き呼び出しボタンが押された場合に1～6の情報を得ることができるかどうかを解説する．参考として，3階建てに4機のエレベータ箱がある場合の，各クラスのインスタンスの関係を示す(**図16**)．

① 実際に呼び出しボタンが押されると，CPUに割り込みが入り「割り込み使用者」として登録されている2階のクラス「上ボ

タン」のインスタンスに通知されることで，使用者が上に行きたいことがわかる

② クラス「上ボタン」はクラス「フロア」とR13で“0..1”対“1”の関係である．通知を受けた「上ボタン」クラスのインスタンスに対してR13で関係する「フロア」クラスのインスタンスは必ず存在し，かつ一意に決定することができる．よって，その「フロア」クラスのインスタンスの属性であるIDから階数を知ることができる

③ ②で一意に決定されたクラス「フロア」のインスタンスとR27で関係する「停止階」に止まる「箱」をR12で知り，R24，R15で現在階を特定することができる．次の箱インスタンスは，そのフロアにある停止階をR17で検索し，各停止階に対してR12で知ることができる．以上をR17の関係がなくなるまで繰り返すことで，すべての箱の現在階を知ることが可能となる

④ すべての箱インスタンスの検索方法は，③と同じ．すべての箱インスタンスに対して，R25の関係先が存在するかしないかで，どこかに向かおうとしているか，していないかは判別できる

⑤ ④のR25，R15で次に止まる階の階数がわかる

⑥ ⑤のR25で得られた次に止まる階については，R26の関係で調べることができる

ここまでに得られた情報を元に，向かわせるエレベータ箱を決定する．後は，決定されたエレベータ箱が移動中でなければ，現在停止階と目標停止階のインスタンスの属性である「高さ」の差をもって，A社製モータを起動するだけである．モータが停止したときには目標階についていることになる．後は，R11で関係している扉を開けて，目標停止階の上向き到着ランプを点滅させれば完了である．ここまでの検証で，ユースケース「呼び出す」のシナリオは，実現可能であるということができた．

2.5 実装

クラスの実装例を示す．簡単のため次に示すクラスを実装した．クラス「エレベータ箱」をBox.h（**リスト1**）およびBox.cpp（**リスト2**）に記した．クラス「ボタン」をButton.h，クラス「上向き呼び出しボタン」をUpButton.hおよびUpButton.cpp，クラス「下向き呼び出しボタン」をDownButton.hおよびDownButton.cpp，クラス「行き先階ボタン」をFloorButton.hおよびFloorButton.cpp，クラス「停止階」をStopFloor.h，クラス「フロア」をfloor.h，クラス「扉」をDoor.hおよびDoor.cppに記した（上記コードはすべて本号付属CD-ROM InterGiga No.30に収録．誌面の都合で**リスト1**，**リスト2**のみ掲載）．

属性名にはコメント，関係は**図14**の関係の番号（R**）と同じにしている．各ボタンクラスは，ボタンが押されたときのアクションをvoid opPush(void)で記述している．扉クラスは，開け閉めおよび扉が閉じた時のアクションをそれぞれvoid opOpen(void)，void opClose(void)，void notifyMotorStop(void)で記述し，エレベータ箱クラスは，停止した時のアクションをvoid notifyMotorStop(void)に記述した．それぞれのアクションは，ソースコードにコメントとして

〔**リスト1**〕**Box.h**

```
#ifndef BOX_H
#define BOX_H

#include "MotorUser.h"

class StopFloor;
class Door;

class Box: public MotorUser { // 箱クラス
private:
    int id;
    Door *door_R11;
    StopFloor *stopFloor_R24; // 現在階
    StopFloor *stopFloor_R25; // 向かっている階
public:
    Box(void) {
        setStopFloor_R24(0);
        setStopFloor_R25(0);};
    Box(int id) {
        setId(id);
        setStopFloor_R24(0);
        setStopFloor_R25(0);
    };
    void notifyMotorStop(void);
    void setId(int val) {
        id = val;
    };
    int getId(void) {
        return id;
    };
    void setDoor_R11(Door *val) {
        door_R11 = val;
    };
    Door *getDoor_R11(void) {
        return door_R11;
    };
    void setStopFloor_R24(StopFloor *val) {
        stopFloor_R24 = val;
    };
    StopFloor *getStopFloor_R24(void) {
        return stopFloor_R24;
    };
    void setStopFloor_R25(StopFloor *val) {
        stopFloor_R25 = val;
    };
    StopFloor *getStopFloor_R25(void) {
        return stopFloor_R25;
    };
};

#endif
```

〔**リスト2**〕**Box.cpp**

```
#include <stdio.h>
#include "Door.h"
#include "StopFloor.h"
#include "Floor.h"
#include "Box.h"

void Box::notifyMotorStop(void) {
    printf("%d号機が%d階に到着した¥n",
        getId(), getStopFloor_R25()->getFloor_R15()->getFloorNum());

    setStopFloor_R24(getStopFloor_R25()); // 到着した階を現在階に設定
    setStopFloor_R25(getStopFloor_R24()->getNextStopFloor_R26());
                                                        // 次に止まる階を設定
    if (getStopFloor_R25()) // 次に向かう階があれば、その前に止まる予定の階をリセット
        getStopFloor_R25()->setPrevStopFloor_R26(0);

    getDoor_R11()->opOpen(); // ドアを開ける
}
```

記述した．

次に，インスタンスの実装例を示す．2種類の実装例を示す．一つは，3階建てに4機のエレベータ箱があり，すべての箱がすべての階に停止する場合．もう一つは，5階建てに2機のエレベータ箱があり，1機は1階，2階，3階に停止し，もう1機は1階，4階，5階に停止する．前者をmain.cpp(**リスト3**)に，後者をmain2.cpp(**リスト4**)に記した．

それぞれのmain関数の最後で，各エレベータ箱の現在階を設定し，その後に例にならって使用者のアクションを記述して実行すると，コンソール箱にシステムの動きが表示される．Microsoft Visual C++6.0以上の環境を使って試してみてほしい．

おわりに

本システムは，エレベータ箱と扉をモータで動かし，アクタの要求に応えるシステムである．世の中にモータは多数あるが，ここでは一般にOA機器などで使用されているステッピングモータを例にあげた．クラス図の**図8**で示されるクラス「A社製モータ」とクラス「A社製モータ使用者」の関係および**図14**のクラ

〔リスト3〕main.cpp

```
#include "Box.h"
#include "BottomUpButton.h"
#include "MiddleUpButton.h"
#include "TopDownButton.h"
#include "MiddleDownButton.h"
#include "FloorButton.h"
#include "Floor.h"
#include "StopFloor.h"
#include "Door.h"

// インスタンス生成
Box box1(1),        // 1号機
    box2(2),        // 2号機
    box3(3),        // 3号機
    box4(4);        // 4号機

Door door1(&box1), // 1号機の扉
     door2(&box2), // 2号機の扉
     door3(&box3), // 3号機の扉
     door4(&box4); // 4号機の扉

Floor floor1(1),    // 1階
      floor2(2),    // 2階
      floor3(3);    // 3階

StopFloor stopFloor1_1(0, &box1, 0, &floor1),              // box1の止まる1階
          stopFloor2_1(10, &box1, 0, &floor2),             // box1の止まる2階
          stopFloor3_1(15, &box1, 0, &floor3),             // box1の止まる3階
          stopFloor1_2(0, &box2, &stopFloor1_1, &floor1),  // box2の止まる1階
          stopFloor2_2(10, &box2, &stopFloor2_1, &floor2), // box2の止まる2階
          stopFloor3_2(15, &box2, &stopFloor3_1, &floor3), // box2の止まる3階
          stopFloor1_3(0, &box3, &stopFloor1_2, &floor1),  // box3の止まる1階
          stopFloor2_3(10, &box3, &stopFloor2_2, &floor2), // box3の止まる2階
          stopFloor3_3(15, &box3, &stopFloor3_2, &floor3), // box3の止まる3階
          stopFloor1_4(0, &box4, &stopFloor1_3, &floor1),  // box4の止まる1階
          stopFloor2_4(10, &box4, &stopFloor2_3, &floor2), // box4の止まる2階
          stopFloor3_4(15, &box4, &stopFloor3_3, &floor3); // box4の止まる3階

BottomUpButton up1(&floor1);                 // 1階の上向き呼び出しボタン
MiddleUpButton up2(&floor2);                 // 2階の上向き呼び出しボタン

MiddleDownButton down2(&floor2);             // 2階の下向き呼び出しボタン
TopDownButton down3(&floor3);                // 3階の下向き呼び出しボタン

FloorButton floorButton1_1(&stopFloor1_1), // box1の1階行き先階ボタン
            floorButton2_1(&stopFloor2_1), // box1の2階行き先階ボタン
            floorButton3_1(&stopFloor3_1), // box1の3階行き先階ボタン
            floorButton1_2(&stopFloor1_2), // box2の1階行き先階ボタン
            floorButton2_2(&stopFloor2_2), // box2の2階行き先階ボタン
            floorButton3_2(&stopFloor3_2), // box2の3階行き先階ボタン
            floorButton1_3(&stopFloor1_3), // box3の1階行き先階ボタン
            floorButton2_3(&stopFloor2_3), // box3の2階行き先階ボタン
            floorButton3_3(&stopFloor3_3), // box3の2階行き先階ボタン
            floorButton1_4(&stopFloor1_4), // box4の1階行き先階ボタン
            floorButton2_4(&stopFloor2_4), // box4の2階行き先階ボタン
            floorButton3_4(&stopFloor3_4); // box4の3階行き先階ボタン

void main(void) {
    // 関連付け
    // ボックスと扉の関係
    box1.setDoor_R11(&door1);
    box2.setDoor_R11(&door2);
    box3.setDoor_R11(&door3);
    box4.setDoor_R11(&door4);
    // フロアクラスのR16(上下階)の関係
    floor1.setUpFloor_R16(&floor2);
    floor1.setDownFloor_R16(0);
    floor2.setUpFloor_R16(&floor3);
    floor2.setDownFloor_R16(&floor1);
    floor3.setUpFloor_R16(0);
    floor3.setDownFloor_R16(&floor2);
    // フロアクラスと停止階のR27(一番端の停止階)の関係
    floor1.setStopFloor_R27(&stopFloor1_4);
    floor2.setStopFloor_R27(&stopFloor2_4);
    floor3.setStopFloor_R27(&stopFloor3_4);

    // 初期設定
    // 各ボックスの現在位置を設定
    box1.setStopFloor_R24(&stopFloor1_1); // box1の現在階1階
    box2.setStopFloor_R24(&stopFloor2_2); // box2の現在階2階
    box3.setStopFloor_R24(&stopFloor1_3); // box3の現在階1階
    box4.setStopFloor_R24(&stopFloor1_4); // box4の現在階1階

    // エレベーター使用者のアクション
    floorButton2_1.opPush(); // 1号機の2階行きボタン押下
    floorButton3_1.opPush(); // 1号機の3階行きボタン押下
    down3.opPush();          // 3階の下向き呼び出しボタン押下
    floorButton1_1.opPush(); // 1号機の1階行きボタン押下
}
```

〔リスト4〕main2.cpp

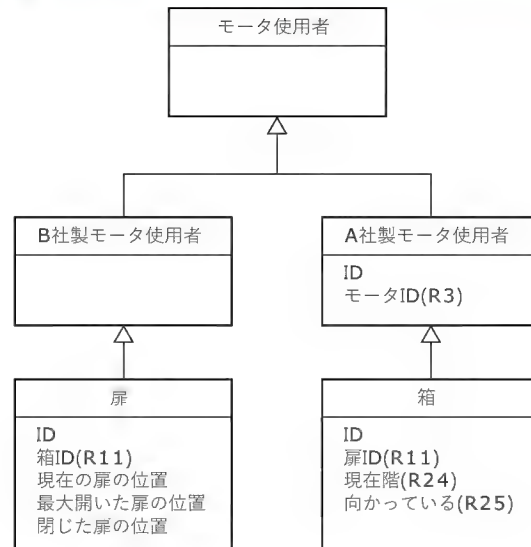
```
#include "Box.h"
#include "BottomUpButton.h"
#include "MiddleUpButton.h"
#include "TopDownButton.h"
#include "MiddleDownButton.h"
#include "FloorButton.h"
#include "Floor.h"
#include "StopFloor.h"
#include "Door.h"

// インスタンス生成
Box box1(1),        // 1号機
    box2(2);        // 2号機

Door door1(&box1), // 1号機の扉
     door2(&box2); // 2号機の扉

Floor floor1(1),    // 1階
      floor2(2),    // 2階
      floor3(3),    // 3階
      floor4(4),    // 3階
      floor5(5);    // 3階

StopFloor stopFloor1_1(0, &box1, 0, &floor1),            // box1の止まる1階
          stopFloor2_1(10, &box1, 0, &floor2),           // box1の止まる2階
          stopFloor3_1(15, &box1, 0, &floor3),           // box1の止まる3階
          stopFloor1_2(0, &box2, &stopFloor1_1, &floor1), // box2の止まる1階
          stopFloor4_2(10, &box2, 0, &floor4),           // box2の止まる4階
          stopFloor5_2(15, &box2, 0, &floor5);           // box2の止まる5階

BottomUpButton up1(&floor1);                             // 1階の上向き呼び出しボタン
MiddleUpButton up2(&floor2);                             // 2階の上向き呼び出しボタン
MiddleUpButton up3(&floor3);                             // 3階の上向き呼び出しボタン
MiddleUpButton up4(&floor4);                             // 4階の上向き呼び出しボタン

MiddleDownButton down2(&floor2);                         // 2階の下向き呼び出しボタン
MiddleDownButton down3(&floor3);                         // 3階の下向き呼び出しボタン
MiddleDownButton down4(&floor4);                         // 4階の下向き呼び出しボタン
TopDownButton down5(&floor5);                            // 5階の下向き呼び出しボタン

FloorButton floorButton1_1(&stopFloor1_1),               // box1の1階行き先階ボタン
            floorButton2_1(&stopFloor2_1),               // box1の2階行き先階ボタン
            floorButton3_1(&stopFloor3_1),               // box1の3階行き先階ボタン
            floorButton1_2(&stopFloor1_2),               // box2の1階行き先階ボタン
            floorButton4_2(&stopFloor4_2),               // box2の4階行き先階ボタン
            floorButton5_2(&stopFloor5_2);               // box2の5階行き先階ボタン

void main(void) {
    // 関連付け
    // ボックスと扉の関係
    box1.setDoor_R11(&door1);
    box2.setDoor_R11(&door2);
    // フロアクラスのR16(上下階)の関係
    floor1.setUpFloor_R16(&floor2);
    floor1.setDownFloor_R16(0);
    floor2.setUpFloor_R16(&floor3);
    floor2.setDownFloor_R16(&floor1);
    floor3.setUpFloor_R16(&floor4);
    floor3.setDownFloor_R16(&floor2);
    floor4.setUpFloor_R16(&floor5);
    floor4.setDownFloor_R16(&floor3);
    floor5.setUpFloor_R16(0);
    floor5.setDownFloor_R16(&floor4);
    // フロアクラスと停止階のR27(一番端の停止階)の関係
    floor1.setStopFloor_R27(&stopFloor1_2);
    floor2.setStopFloor_R27(&stopFloor2_1);
    floor3.setStopFloor_R27(&stopFloor3_1);
    floor4.setStopFloor_R27(&stopFloor4_2);
    floor5.setStopFloor_R27(&stopFloor5_2);

    // 初期設定
    // 各ボックスの現在位置を設定
    box1.setStopFloor_R24(&stopFloor1_1);               // box1の現在階1階
    box2.setStopFloor_R24(&stopFloor1_2);               // box2の現在階1階

    // エレベーター使用者のアクション
}
```

ス「A社製モータ使用者」と，クラス「扉」，クラス「箱」のスーパークラスの関係でエレベータ制御システムの制御と各エレベータおよび扉の位置制御を行うモータ制御部分を切り分けることができる．

したがって，モータ制御モデルの開発者は，使用者が誰かということを意識せずに開発を行うことができる．たとえば，扉のモータをA社製からB社製に変更した場合，B社製モータ制御モデルの開発者は，B社製モータ制御のためのモデルを新たに作成し，B社製モータを実際に使用する扉は**図17**で示すように変更すればよいわけである．

また，エレベータ制御システムに拡張があった場合にどのようになるかについては，実装ソースコード内に記した．モデルへの変更はなく，インスタンスの増減，および関係の変更のみで対応可能なことがわかる．ソースコード上で実際にどのようになるかは，実装例のmain.cppとmain2.cppを見比べてみてほしい．

今回のエレベータ制御システムの開発では，要求を実現するモデルを構築する過程を詳細に考察したが，制御対象の特徴を吟味し，そのモデルにそのつど反映させながら進めることにより，結果として物理的に存在する部分の関連付けだけで，制御モデルを作成することができた．

途中に作成した関連を示すモデル（**図8**，**図9**，**図13**）は，開発対象となるドメインに応じて，ユースケースを補足する形で必要になるものである．今回はシーケンス図などの中間成果物は割愛したが，シナリオとクラス図が準備できれば，シーケンス図の作成は難しいことではない．

〔図17〕モータ使用者の関係

のはら・ありと/すぎうら・ひでき　富士ゼロックス(株)

第3章

ユースケース駆動の開発手順にしたがって実際に組み込みソフトを作成してみる

電波時計システムモデルの検証

オブジェクト指向を導入した開発手法の一つに「ユースケース駆動」がある．これは，ユースケースを使ってシステム分析し，ユースケースからクラスを抽出し，そのクラスでユースケースが実現できることを動的なモデルで検証する手法である．本章では，このユースケース駆動を使って実際に電波時計モデルを実装する．作成するドキュメント類を中心に解説するが，電波時計システムの実装コードを本号付属CD-ROM InterGiga No.30に収録したので，こちらも参考にしていただきたい．（編集部）

1 ユースケースの実力

ユースケースによりシステムを分析し，作成したユースケースからクラスを抽出して，そのクラスでユースケースが実現できることを動的なモデルで検証するというのが，一般的に**ユースケース駆動**と呼ばれる開発方法である．ここでは，実際に開発するにあたり，ユースケースをどのように抽出すべきか，また，ユースケース以外で組み込みシステムを分析・設計するにあたって必要な情報について，電波時計システムの例をとおして確認する．

さらに，ユースケースを元に電波時計システムを動作させるための組み込みソフトを，ユースケース駆動の開発手順にしたがって実際に作成し，現実の製品開発で必要になる資料について考察する．

1.1 コンセプトペーパーからユースケースを抽出

オブジェクト指向で初めて開発を行うにあたり，具体的に何から行えばよいのだろうか？

いきなりオブジェクト指向で開発しようと考えて，誰でも簡単に始めることができるだろうか？　UMLを使ってオブジェクト指向で設計しようといっても，まず何をやったらよいのか，どう分析したらよいのかがわからないと思う．

設計が終了した段階で，開発したものが正しくできているのか，設計方法は実装に有効なのか，ということがわからないのでは不安になるだけだ．不安な思いを抱えたまま，あいまいに開発をすすめても時間がかかるし，開発を楽しめない．採用した技術が安全で確実であるという自信をもってすすめられる開発により近づくためには，どうしたらよいのだろうか？

ここでは，電波時計を題材に，電波時計システムの分析・設計を実際に行う．手始めに，分析としてユースケースの作成，ユースケースシナリオの作成，クラス図の作成を順に行うことにする．

まず，ユースケースを作るにあたり，何から手をつけたらよいのだろうか？　何がユースケースでアクタは何なのか，どうやってユースケースを抽出したらよいのだろうか？

ユースケースの検討ばかりしていても何も始まらないので，まず開発対象システムのコンセプトペーパーを作成することにした．システムに求められていることは何か，どういったことができるシステムなのかをブレーンストーミングの要領で挙げてみる．

電波時計システムは，時刻を表示していて，アラームがあって，表示日時を設定する機能があって……と考えはじめると，機能を中心にした項目の列挙になってしまう．

機能項目の列挙では，構造化分析手法と変わらず，本来のオブジェクト指向の開発とは違うのではないかという疑問が生じた．イメージとしては，システムが「どういう機能をもっているか」ではなく，システムは「何ができるのか」を並べてユースケースにつなげたい．実際に作業してみると，開発側の人間は意外とユーザーとしての立場になることが難しいようだ．そこで，少し視点を変えて，取り扱い説明書をイメージしてみた．自分が一人のユーザーとなって，具体的に操作してできることをイメージした要求仕様から洗い出す．電波時計システムのどこをどう操作したら何ができるのかを列挙していく（**図1**）．

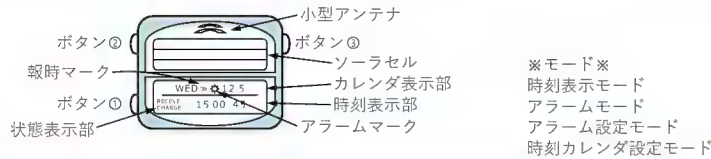
ボタン①を押したらアラーム設定が開始される，ボタン②を押したら受信状態が表示される，というような操作手順を要求仕様のまま列挙して，ユースケースの候補とする．

1.2 どうやってユースケースを抽出したか

分析ユースケース作成の段階では，前述のコンセプトペーパーで列挙した「ユーザーが操作してできること」をユースケースの候補として抽出した．コンセプトペーパーで取り上げた一つ一つの項目に対し，“誰”が“何”をできるのか検証する．“誰が”が主アクタで，“できること”がユースケースになる．“できること”を行うときに，分析対象のシステムがもっていないものが必要になれば，副アクタとする．コンセプトペーパーに抽出した

〔図1〕 コンセプトペーパー(1)「システムができること」の列挙

(1) 概観

※モード※
時刻表示モード
アラームモード
アラーム設定モード
時刻カレンダ設定モード

(2) 操作と操作によって実現されること

		操作[ボタンの組み合わせ]	実現されること	詳　細
1	共通	アラーム音が鳴っている時に①または②または③を押下	アラーム音を停止する	鳴っているアラームを止める(停止されない場合は20秒経過で自動停止する)
2		①と②と③を2秒以上押下	システムをリセットする	システムをリセットする．押下している間，表示が消える
3		標準時刻電波受信中に①または②または③を押下	受信動作中断を行う	標準時刻電波受信中に，受信動作を中止して時刻表示モードになる
4	時刻表示モード	ボタン①を押下	アラームの設定を開始する	時刻表示モードで押下するとアラームモードになる
5		①と②を押下	アラーム音の試し聴きを行う	アラーム音が鳴る
6		ボタン③を2秒以上押下	時刻・カレンダを設定する	時刻カレンダ設定モードになる
7		ボタン②を押下	受信状態の表示を行う	カレンダ表示部に受信できなかった日数を表示する
8		ボタン②を2秒以上押下	標準時刻電波の強制受信を行う	「RECEIVE」と表示して受信状態のドット表示を行う．受信に成功すると受信成功音が鳴り，受信時刻を表示する．失敗すると受信エラー音が鳴り「ERR」と表示する
9	アラームモード	ボタン①を押下	日時・時刻を表示する	アラームモードで押下すると時刻表示モードになる
10		ボタン③を2秒以上押下	アラーム時刻の設定を開始する	アラーム設定モードになる
11		ボタン②を押下	アラームを鳴らす，または鳴らさないの選択をする	押下ごとにアラームマークが表示(鳴らす)→非表示(鳴らさない)になる
12		アラーム時刻設定中にボタン②を押下	アラーム時/分の設定を開始する	ボタン②を押下するごとに時または分が交互に設定可能となる
13		アラーム時/分の設定中にボタン①を押下	アラーム時/分を入力する	時または分が1ずつ進む
14		アラーム時/分の設定中にボタン①を押しつづける	アラーム時/分を入力する	時または分が早く1ずつ進む
15		アラーム時/分の設定中にボタン③を押下	アラームの時/分の設定を終了する	アラームモードに戻る．アラームマークが点灯する
16	時刻カレンダ設定モード	ボタン②を押下	設定箇所を選択する	報時ON/OFF→秒→分→時→日→月→年→時間表示12H/24Hの順に設定箇所を選択する
17		ボタン①を押下	選択された個所の設定を変更する	設定中の数字が1ずつ進む．数字以外の箇所は選択内容の切り替えを行う
18		ボタン③を押下	時刻カレンダ設定モードを終わる	時刻表示モードに戻る
19		2～3分ボタンを押さない	時刻カレンダ設定モードを終わる	時刻表示モードに戻る
20		ボタン③を2秒以上押下	報時・操作確認音のON/OFF設定を設定する	ONを設定すると報時マークが表示される
21	システムリセット中	押下中のボタンを離す	リセット後設定を開始する	1996が表示され，年の設定を行う状態になる
22		リセット後，表示年が点滅中にボタン①を押下	リセット後の年設定を行う	表示中の年が1ずつ進む
23		リセット後，年を設定したボタン③を押下	リセット後標準時刻電波を受信する	標準時刻電波を受信しリセット終了する

※現在の標準時刻電波のカレンダは1月1日からの積算日である←西暦のみはセットされている必要がある
※電源がなくなった場合は表示が消える．その後電気を与えるとシステムリセット後設定開始になる
※時刻表示は24H表示とする

操作(つまりイベント)によって期待される結果をユースケースとして抽出した．

時計システムでは，ユーザーの視点でコンセプトペーパーを作成したので，操作を行うユーザーを主アクタとしてとらえる．コンセプトペーパーに列挙した内容についてはそれぞれをユースケースとしてとらえ，そのユースケースは何ができるものなのかを考察した．

さらに，取り扱い説明書という視点で要求仕様からもう一歩ユーザーに歩み寄ることで，その操作で行われる内容にユーザーは何を期待しているのかについても考慮する．コンセプトペーパーで挙げた内容のすべてをユースケースとするつもりで抽出作業を進める．順番にみていくと，これらの操作内容は，システムが“できること”と，“それに付随してできること”に分かれてこないだろうか？　つまり，できることの種類が2種類あることになる(**表1**)．

1.3　ユースケースで解釈することが難しい部分

ユースケースを抽出する際の視点の問題は，現実の開発ではしっかり考察しておく必要がある．今回のようにコンセプトペーパーからユースケースの抽出を行っても，“ユースケース抽出の視点”には感覚的にとまどうところがある．どのくらいのまとめ具合(粒度)で一つのユースケースにしたらよいか，まとめる必要があるのか？　細かくしたらすべてをユースケースに抽出してよいとも考えられるし，広い意味では「時計を使う」ということだけがユースケースではないかとも考えられる．

コンセプトペーパーは操作の視点で考えたので，視点の画一性は作られているとして，あまり深く突き詰めすぎないように注意する必要がある．また，常にユースケースの“できること”について着目し，それぞれのユースケースの目的が合致するならユースケースを適度に統合して整理する．ユースケースを整理する際にも注意が必要で，

〔表1〕**コンセプトペーパーから抽出したユースケース**（コンセプトペーパーの抽出項目×ユースケース）

	操作［ボタンの組み合わせ］	実現されること	できること	ユースケース
1	アラーム音が鳴っている場合に①または②または③を押下	アラーム音を停止する	ユーザーはアラーム音を停止できる	アラームに関する操作
2	①と②と③を2秒以上押下	システムをリセットする	ユーザーはシステムをリセットできる	ユーザーはシステムをリセットできる
3	標準時刻電波受信中に①または②または③を押下	受信動作中断を行う	ユーザーは時刻電波受信を中断できる	ユーザーは時刻電波受信を中断できる
4	時刻表示モードにてボタン①を押下	アラームの設定を開始する	ユーザーはアラームの設定を開始できる	アラームに関する操作
5	時刻表示モードで①と②を押下	アラーム音の試し聴きを行う	ユーザーはアラーム音を聞くことができる	
6	時刻表示モードでボタン③を2秒以上押下	時刻・カレンダを設定する	ユーザーはカレンダの設定を開始できる	ユーザーはカレンダを設定できる
7	時刻表示モードにてボタン②を押下	受信状態の表示を行う	ユーザーは時刻電波受信が正常に行われたか確認できる	表示されている時刻が調整された時刻か確認できる
8	時刻表示モードにてボタン②を2秒以上押下	標準時刻電波の強制受信を行う	ユーザーは標準時刻電波の受信を行える	時刻を調整することができる
9	アラームモードにてボタン①を押下	日時・時刻を表示する	ユーザーは時刻を知る	ユーザーは時刻を知る
10	アラームモードにてボタン③を2秒以上押下	アラーム時刻の設定を開始する	アラーム時刻の設定を開始できる	アラームを設定できる
11	アラームモードにてボタン②を押下	アラームを鳴らすまたは鳴らさないの選択をする	アラーム設定を選択できる	10で抽出した"アラームを設定できる"ユースケースの中で実現する
12	アラーム時刻設定中にボタン②を押下	アラーム時/分の設定を開始する	アラーム時刻の設定を開始できる	
13	アラーム時/分の設定中にボタン①を押下	アラーム時/分を入力する	設定する時または分を1進めることができる	
14	アラーム時/分の設定中にボタン①を押しつづける	アラーム時/分を入力する	設定する時または分を早く1進めることができる	
15	アラーム時/分の設定中にボタン③を押下	アラームの時/分の設定を終了する	アラーム時刻の設定を終了できる	
16	時刻カレンダ設定モードにてボタン②を押下	設定箇所を選択する	日付設定箇所を選択できる	6で抽出した"カレンダを設定できる"ユースケースの中で実現する
17	時刻カレンダ設定モードにてボタン①を押下	選択された箇所の設定を変更する	選択された箇所の設定を変更できる	
18	時刻カレンダ設定モードにてボタン③を押下	時刻カレンダ設定モードを終わる	時刻・カレンダを設定できる	
19	時刻カレンダ設定モードにて2～3分ボタンを押さない	時刻カレンダ設定モードを終わる	時刻・カレンダを設定できる	
20	時刻カレンダ設定モードにてボタン③を2秒以上押下	報時・操作確認音のON/OFF設定を設定する	報時・操作音を設定できる	
21	（ボタン①②③を同時に押下したあと）押下中のボタンを離す	リセット後設定を開始する	リセット後設定を開始できる	2で抽出した"システムをリセットできる"ユースケースのなかで実現する
22	リセット後，表示年が点滅中にボタン①を押下	リセット後の年設定を行う	設定する年を変更できる	
23	リセット後，年を設定した後ボタン③を押下	リセット後標準時刻電波を受信する	標準時刻電波を受信し時刻を調整できる	

- "できること"が違う場合には統合しない
- "できること"は違うが目的が同じならば統合する

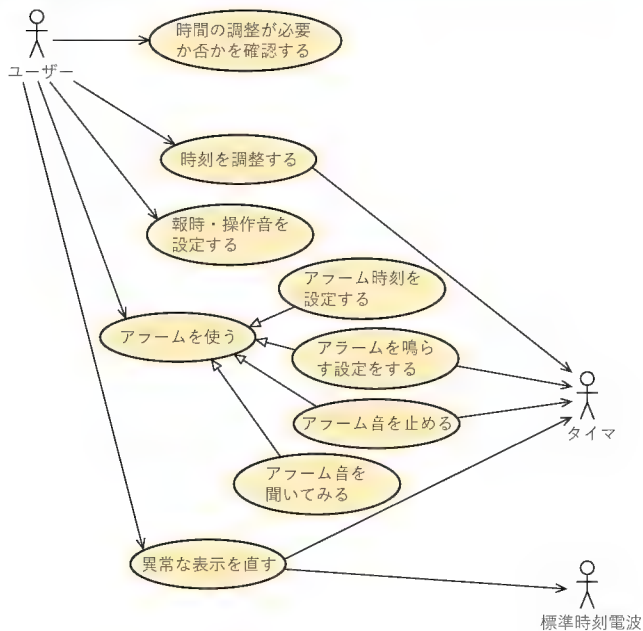
という方針で作業を進めるとよい．

ユーザー操作で時刻電波受信を開始して時刻を調整することと，時刻カレンダ設定モードで時間を設定できることは，操作は違うが目的は同じと考えて「時刻を調整する」というユースケースに統合した．「ユーザーはアラームを設定できる」，「ユーザーはアラーム音を聞くことができる」のようなアラームに関するユースケースは，それぞれ見かけ上行われることは異なるが，じつは総じてアラームを使っていると考えられるので，「アラームを使う」という抽象ユースケースを作成して，ユースケースの関係を示すことにした（**図2**）．

1.4 システムが本来もっている外部との同期をともなうサービス

電波時計システムでは，Power On後，定期的に時報との同期を行うが，これをどう表現すればよいのか？ ユーザーの立場に立って，ユーザーの操作に着目してユースケースを抽出したわけだが，はたして操作によって結果がもたらされることだけですべてのユースケース抽出ができたのだろうか？

たとえば，システムが見かけ上自律して処理していること，つまりアクタの刺激を受けずに"自動"で行っていることがあれば，それもシステムが"できること"としてユースケースに抽出する必要があるのではないだろうか？ 新たにこのような視点に立ち，ユーザー操作で実行されること以外にシステムがしている

〔図2〕分析ユースケース図(1)

ことはないかを考察してみる．コンセプトペーパーに，ユーザーの操作以外でシステムが行うことを追加する(**図3**)．

電波時計システムにおいては，ユーザーが操作・設定して使用するほかに，定期的に標準時刻電波を受信して内部時刻を自動調整する仕様になっている．これはどうユースケースに表現したらよいだろうか？

システムが“できること”なので，システムは「時刻を自動調整することができる」ということで，「時刻を自動的に調整する」というユースケースとして抽出した．時刻を調整することで誰が何をしたいのか，できるかと考えると，「ユーザー」が「調整された時刻を知る」ことができるということになる．システムからのサービスはユーザーが利用するものだから，主アクタはユーザーとも考えられるが，ユーザーの介入がないこのユースケースは，ユーザーの介入する操作とは区別して表現できないだろうか？

主アクタをユーザーとしたユースケースでは，アクタのボタン操作がイベントとして各サービスのトリガになっている．このことから考えてみると，システムが自動的に行うサービスのトリガ(以下，自動トリガ)は何になるだろうか．「自動的に……」ということを表現しようとすると，やはり実装する言語で実装のしくみを考えたくなる．ところが，分析の段階では自動トリガをソフトウェアで実装するのか，ハードウェアで実装するのかまでは決まっていない．システム分析を行っているような場合には，分析の結果からソフトウェア，ハードウェアのどちらで実装したらよいかを決める場合もある．今回の電波時計システムの例では，自動トリガはシステム外部にハードウェアのしくみとしてあることを前提として，作業を進めることにする．

電波時計システムでは，システムが定期的に自動で行う処理を外部アクタとしてのタイマがコンテキストとしてのシステムに

〔図3〕コンセプトペーパー(2)「自動で行っていること」

	操作ボタンの組み合わせ	実現されること	詳細
24	標準時刻電波受信を行う	毎日午前2：00に標準時刻電波の受信を行い，時刻を調整する	毎日午前2：00に受信を開始する．正常に受信できない場合は5回までリトライする
25	時刻を表示	時刻を更新し表示している	――

イベントトリガを与えることを前提に考えた．つまり，タイマを副アクタとして「時刻を自動的に調整する」というユースケースを抽出した．

もう一つ，電波時計システムの場合で考えにくいと感じるのは，システムが「時刻を絶えず表示している」ことである．時計機能としてシステムは常に時間を表示しているが，何のイベントトリガもないままに時刻を表示しているので，アクタがみつからない．時計の役割を素直に考えると，われわれは時計を見ることで現在の時間(らしきもの)を知ることができる．分析の段階では時刻を表示していることで知りたい時間がわかるというユーザーを主アクタとし，“時刻を知る”というユースケースを考えることで，システムの普遍的な役割を表現した(**図4**，**図5**)．

1.5 ユースケース図をどう作るか

ここでは，ユースケース図作成の簡単な説明と陥りやすい間違い，作るうえでわかりにくい部分などのポイントを整理する．

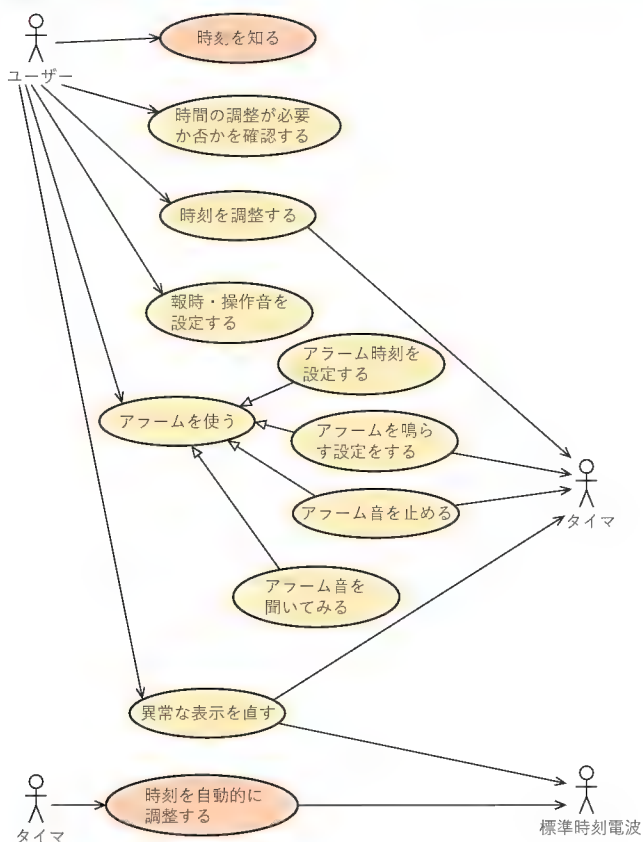
ユースケース図を作成するときに難しいと感じるのは，まず何をユースケースにしたらよいかということである．諸兄により作成されたユースケース図を見ると，「そういうものか」とあいまいに納得できたりするのだが，教科書の例や先輩諸兄の作成したユースケースを参考にして実際の商品開発に置き換えようとすると，とたんにどうユースケースを抽出したらよいのかがわからなくなってしまう．ソフトウェア開発経験があると，要求されたことがソフトウェアのしくみや構造に直結してしまうので，システムを「どうやって動かそうか」という考えになりやすい．

ユースケース図の作成というよりは，ユースケースの抽出において重要なポイントは，

- 何をユースケースにするか
- 主アクタは何か
- 副アクタは何か

ということに着目し，これらを具体的にしていくことといえる．

分析のユースケースではシステムのしくみ，つまり，どうやって実現するかということを考えないように注意して，「システムをどう使うのか？」，「システムで何ができるのか？」を考えるようにする．「何ができるのか？」と考えるということは，「システムが実現できること」，つまり使用者がシステムに期待することを考えていることになる．さらに，「それは誰が使うのか？」と考えたときの“誰”を主アクタとすることで，「システムが実現できること」を主アクタ側から見直して，ユースケースにしていく．結果として，アクタが“できること”をユースケースとして抽出

〔図4〕分析ユースケース図(2)

することになる．ユースケースを実現するときにシステムのコンテキスト外の制御要素が必要であれば，すべて副アクタとして抽出する．

電波時計システムの場合は，コンセプトペーパーを作ることで機能分割的な考えと分析ユースケース抽出の関係を知ることができた．考え方のパターンを作ることでユースケースが考えやすくなるのはたしかである．あとは，抽出に行き詰ったときや表現しにくいことがある場合に，そのつどシステムのしくみ，つまり実現手段にとらわれていないことを注意深く確認したうえで，目的とそのユースケースが必要な理由を検討してユースケースを抽出すればよい．この段階でユースケースシナリオを書き始め，目的やそのユースケースが必要な理由について記述すればよい．

今回は，ユースケースを整理する段階で，抽象ユースケースという考え方を使ったが，実際の製品開発でこの考え方を利用する際には，ユースケース図を見たときに要求の整理ができているということはわかるものの，シナリオ作成時には，抽象ユースケースのシナリオに何を書くべきか，また，抽象ユースケースから使用≪uses≫，拡張≪extend≫されるユースケースシナリオの主アクタや副アクタをどうすればよいか，事前条件・事後条件の抽象ユースケースの関係はどうなるかといった問題を解決しておく必要がありそうだ．

2 誰もがわかるモデルとその課題

2.1 コンセプトペーパーによるモデリング

ユースケースから作るより，コンセプトペーパーから作ったほうが，モデルとしては素直になるのではないだろうか？　電波時計システムの例を通して，モデル(クラス図)を解説，検証する．

分析ユースケースができたところで次に，電波時計システムのクラス図を作成していく．先に分析ユースケースを作成したのだから分析ユースケースから分析クラス図を作成したいところだが，分析ユースケースを前にしても，クラス図作成の方法はわからない．「クラス図ってどうやって書くの？」という疑問を抱えたまま，参考になる資料や教科書を探して調べてみると，ユースケース図からクラス図への移行作業については思いのほかあっさり書かれていることが多く，どこをどう参考にしてよいものかわからないのが現実である．

ここでもう一度確認しておくべきことは，**分析ユースケースはシステムのもつ目的を確認するもの**ということだ．システムに求められる「事(こと)」を分析ユースケースで確認し，それをどのような構成で実現しようとしているかを分析クラス図で表したい．

分析の段階では，コンセプトペーパーからハードウェアの仕様を洗い出し，ソフトウェアで何を制御するのか考える．コンセプトペーパーには，(1)概観，(2)操作と操作によって実現されることを書いた．概観には時計の部品を書いたので，ここからシステムが制御する物を洗い出し，まずは部品ごとに制御クラスを作る．現物の構成をクラス図にすることで，システムが何を制御しようとしているかがわかりやすくなる．電波時計システムでは，ボタンがユーザーインターフェースになるとか，ディスプレイがあってデータが表示されている……ということがモデルを見ただけでわかる．この現物の構成で，分析ユースケースそれぞれを実現できるように，分析クラス図に操作を記入していく(**図6**)．

できあがった分析クラス図のモデル検証は，シーケンス図を作って行う．シーケンス図でクラス間のインターフェースをユースケースシナリオどおりに進めることができるか確認していく．その際に，

- 作成したモデルでシナリオが実現できるか
- 必要なインターフェースは何か

に着目して検証を進める．作成したクラス図でユースケースシナリオが実現できないと気づいた場合は，再度分析クラス図・分析ユースケースの検討を行う．分析ユースケース，分析クラス図と分析シーケンス図の検討を繰り返すことで要求とシステムの整合性を高めていく(**図7**)．

2.2 分析から設計への接続

分析から設計へ至る際に，本当に分析モデル，設計モデルの二つのモデルが必要になるのだろうか？　分析モデルをそのま

〔図5〕分析ユースケースシナリオ

Use Case Document Form　2.0

分析

版No.	発行日	作成者	承認
K1		石田栄子	

分析ユースケース：時刻を調整する

○ **システム**

・電波修正時計システム

○ **アクタ**

・メインアクタ：ユーザー
・サブアクタ：タイマ

○ **目的**

・ユーザーがシステムの時刻を調整する

○ **概要**

・ユーザーはボタン操作により標準時刻電波受信による時刻調整を強制的に行うことができる．正常に電波受信できない場合は時刻カレンダ設定モードにて各項目を入力設定し，時刻を調整する

○ **事前条件**

・システムは時刻表示モードになっている

○ **事後条件**

・システムは時刻の調整が完了し時刻表示モードになっている

○ **シナリオ記述**

シナリオ1．標準時刻電波の強制受信を行う場合
シナリオ2-1．ボタン操作により時刻調整項目の設定を行う場合
シナリオ2-2．時刻調整項目の設定中にユーザーが操作を止めた場合

基本系列

シナリオ1-1．

アクタアクション	システムレスポンス
1．ユーザーはボタン②を2秒以上押す	2．「RECEIVE」マークを点滅表示し，標準時刻電波の受信を開始する
	3．カレンダ表示部に受信状況のドット表示と受信時間を表示する
	4．電波受信動作を終了し受信データが正常であることを確認する
	5．受信データをもとに時刻を調整する
	6．ドット表示部に「RCVD」と表示し受信成功音を鳴らす
	7．5秒経過後時刻表示モードに戻る

シナリオ2-1．

アクタアクション	システムレスポンス
1．ボタン③を2秒以上押す	2．時刻カレンダ設定モードになり「報時」の選択状態を表示する
3．ボタン①を押す	4．「報時」のONまたはOFFを押される度に交互に表示する
5．ボタン②を押す	6．「報時」を表示状態で設定する
	7．時刻・カレンダを現在の設定で表示し「秒」を点滅表示する
8．ボタン①を押す	9．「秒」を一つ進める（押されるたびに一つ進める）
10．ボタン②を押す	11．「秒」を表示中の数字で設定する
	12．「分」を点滅表示する
13．ボタン①を押す	14．「分」を一つ進める（押されるたびに一つ進める）
15．ボタン②を押す	16．「分」を表示中の数字で設定する
	17．「時」を点滅表示する
18．ボタン①を押す	19．「時」を一つ進める（押されるたびに一つ進める）
20．ボタン②を押す	21．「時」を表示中の数字で設定する
	22．「日」を点滅表示する
23．ボタン①を押す	24．「日」を一つ進める（押されるたびに一つ進める）
25．ボタン②を押す	26．「日」を表示中の数字で設定する
	27．「月」を点滅表示する
28．ボタン①を押す	29．「月」を一つ進める（押されるたびに一つ進める）
30．ボタン②を押す	31．「月」を表示中の数字で設定する
	32．「年」を点滅表示する
33．ボタン①を押す	34．「年」を一つ進める（押されるたびに一つ進める）
35．ボタン②を押す	36．「年」を表示中の数字で設定する
※4.～36.を繰り返す	
37．ボタン③を押す	38．時刻表示モードに戻る

シナリオ2-2．［シナリオ2-1.の2.以降のどこでも］

アクタアクション	システムレスポンス
1．タイマはボタンが2分間押されないことを通知する	2．表示中の数字で時刻・カレンダを設定する
	3．時刻表示モードに戻る

代替系列

シナリオ１の代替系列1．標準時刻電波の強制受信を行い，正常に受信ができない場合

アクタアクション	システムレスポンス
1．ユーザーはボタン②を2秒以上押す	2．「RECEIVE」マークを点滅表示し，標準時刻電波の受信を開始する
	3．カレンダ表示部に受信状況のドット表示と受信時間の表示を行う
	4．電波受信動作を終了し受信したデータが正常でないことを確認する
	5．ドット表示部に「ERR」を点滅表示し，受信エラー音を鳴らす
	6．5秒経過後時刻表示モードに戻り，曜日表示部に「ERR」を表示する

シナリオ1の代替系列2．受信動作をユーザーが中断し時刻の調整ができない場合

アクタアクション	システムレスポンス
1．ユーザーはボタン②を2秒以上押す	2．「RECEIVE」マークを点滅表示し，標準時刻電波の受信を開始する
	3．カレンダ表示部に受信状況のドット表示と受信時間の表示を行う
4．ユーザーはボタン①②③のいずれかを押す	5．受信動作を中止する
	6．受信状態表示を終了し，時刻表示モードに戻る

○ **注意点**

・ユーザー操作で項目設定する場合は，秒分時日月年はそれぞれボタン①が押されるたびに表示中の数字から一つ進む．秒・分は59の次は0，時は23の次は0，日は31の次は1，月は12の次は1，年は2045の次は1996となる．ボタン③を押すまで報時→秒→分→時→日→月→年の設定を繰り返す

○ **Note**

・曜日は，年月日よりシステム内で算出するため入力設定はできない

○ **I/F記述**

・とくになし

○ **拡張ポイント**

・とくになし

○ **使用ユースケース**

・とくになし

○ **関連ユースケース**

・とくになし

○ **関連クラス**

・とくになし

○ **補足ダイアグラム**

・とくになし

○ **補足資料**

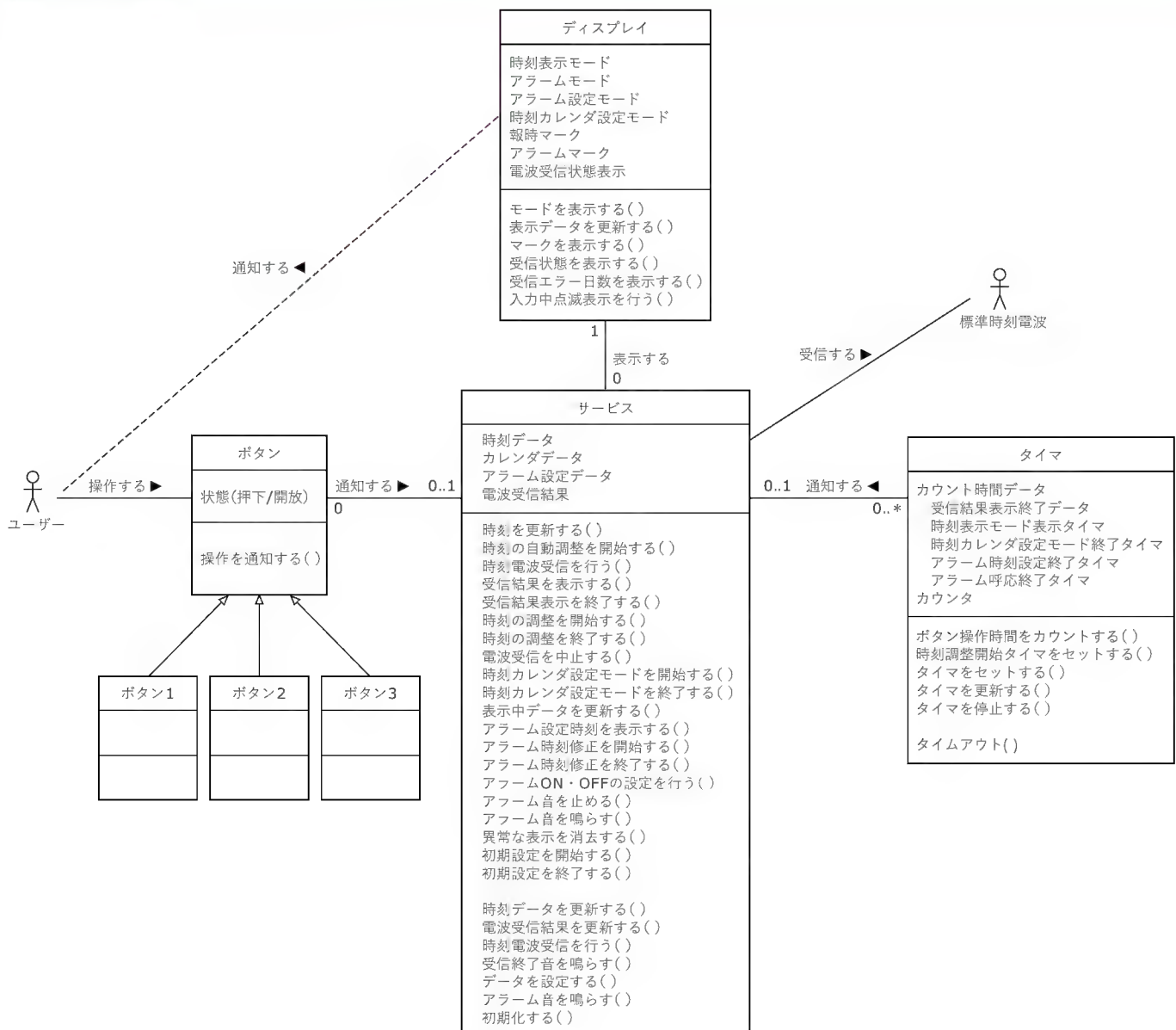
・とくになし

○ **テストでの留意事項**

・とくになし

○ **変更履歴**

改定NO.	変更者	変更日	変更箇所	変更内容	備　考

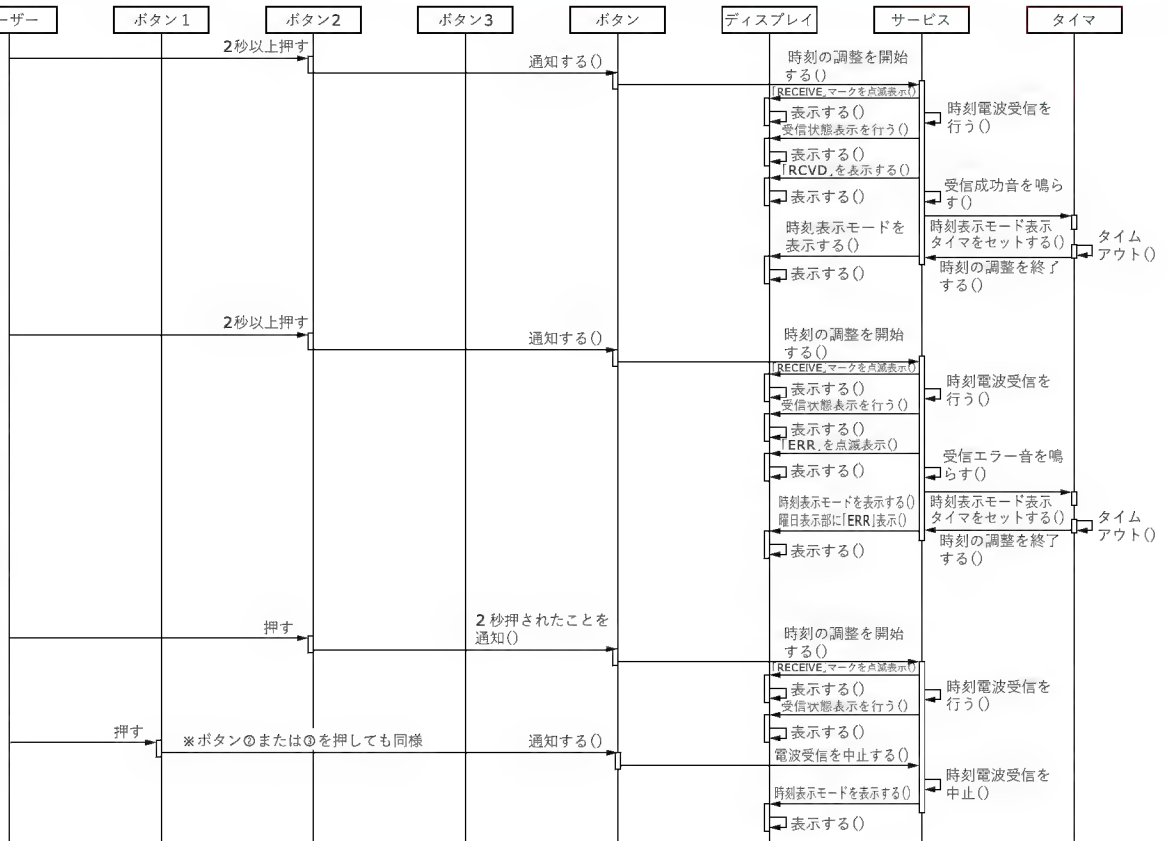
〔図6〕分析クラス図

ま実装することの問題は何か？　つまり，“What”に着目したモデルでそのまま実装しようとすると，どんな問題が起こるのか？について考察する．

分析モデルを作成したが，このモデルからすぐに実装を行えるだろうか．分析モデルでは，「できること」とソフトが制御する対象の特定を中心にシステムを考えてきたが，実際のシステムではどのようにふるまったらよいのだろうか？　システムの目的すなわち，分析ユースケースの“WHAT”と制御対象がわかっても，ソフトウェアが行う処理の境界が明確になっていないため，システムのしくみとしての“HOW”は考えにくい．対象システムで実際に要求にしたがった処理をするソフトウェアを考えると，システムの目的だけを詳細化してもコードを組むことはできない．

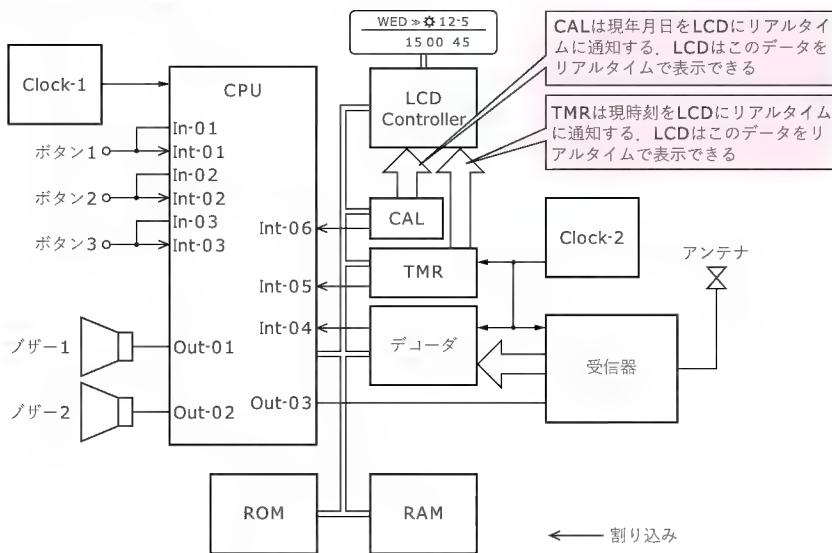
組み込みシステムの開発はソフトウェアとハードウェアの共同開発であり，システムに要求された機能を実現するためにソフトウェア，ハードウェアの双方で役割を分担している．この分担がはっきりしていなければ，どこまでソフトウェアで実現したらよいのかわからない．

たとえば，ソフトウェアがシステムの中で正しく機能するためには，実現手段としてのハードウェアからソフトウェアに要求仕様として示されるもの，具体的には，制御対象についてどのデバイスをどのように使わなければいけないのか，ソフトウェアが自由にふるまってはいけないといった制約などを満たす必要がある．この仕様や制約を満たしてはじめて，組み込みシステムに対する要求を実現できるのである．

組み込みソフトウェアにとっては，実装にともなう制約事項

4. 時刻を調整する
(1/3)
ユーザー
ボタン1
ボタン2
ボタン3
ボタン
ディスプレイ
サービス
タイマ
●シナリオ1 基本系列
ユーザーはボタン②を2秒以上押下するとディスプレイに「RECEIVE」と表示され時刻電波の受信が開始される
受信が正常終了すると受信成功音が鳴り，受信データをもとに時刻が調整され，「RCVD」と表示される
「RCVD」表示から5秒後に時刻表示モードに戻る
2秒以上押す
通知する()
時刻の調整を開始する()
「RECEIVE」マークを点滅表示()
表示する()
受信状態表示を行う()
時刻電波受信を行う()
表示する()
「RCVD」を表示する()
表示する()
受信成功音を鳴らす()
時刻表示モードを表示する()
時刻表示モード表示タイマをセットする()
タイムアウト()
表示する()
時刻の調整を終了する()
●シナリオ1 代替系列1
ユーザーはボタン②を2秒以上押下するとディスプレイに「RECEIVE」と表示され時刻電波の受信が開始される
受信が正常終了しない場合，受信エラー音が鳴り，ドット表示部に「ERR」と点滅表示される
受信動作終了から5秒後に時刻表示モードに戻る．曜日表示部に「ERR」と表示される
2秒以上押す
通知する()
時刻の調整を開始する()
「RECEIVE」マークを点滅表示()
表示する()
受信状態表示を行う()
時刻電波受信を行う()
表示する()
「ERR」を点滅表示()
表示する()
受信エラー音を鳴らす()
時刻表示モードを表示する()
曜日表示部に「ERR」表示()
時刻表示モード表示タイマをセットする()
タイムアウト()
表示する()
時刻の調整を終了する()
●シナリオ1 代替系列2
ユーザーはボタン②を2秒以上押下するとディスプレイに「RECEIVE」と表示され時刻電波の受信が開始される
時刻電波受信中にユーザーがボタンを押下すると，受信動作を中止し時刻表示モードに戻る
押す
2秒押されたことを通知()
時刻の調整を開始する()
「RECEIVE」マークを点滅表示()
表示する()
受信状態表示を行う()
時刻電波受信を行う()
押す
※ボタン②または③を押しても同様
通知する()
表示する()
電波受信を中止する()
時刻電波受信を中止()
時刻表示モードを表示する()
表示する()
4. 時刻を調整する
(2/3)
ユーザー
ボタン1
ボタン2
ボタン3
ボタン
ディスプレイ
サービス
タイマ
●シナリオ2 基本系列1
ユーザーがボタン③を2秒以上押すと時刻カレンダ設定モードになり「報時」の設定状態を点滅表示する
2秒以上押す
通知する()
時刻カレンダ設定モード開始()
時刻カレンダ設定モード終了タイマ設定()
報時操作音の設定表示()
確認する
表示する()
ユーザーがボタン②を押すと時刻カレンダ入力表示になり「秒」を点滅表示する
押す
通知する()
「秒」設定開始()
タイマ更新()
時刻・カレンダ入力表示()
報時操作音を設定する()
現在の設定を確認する
表示する()
ボタン①を押すと，点滅中の数字が1進む
押す
通知する()
表示中の「秒」を1進める()
タイマ更新()
時刻・カレンダ入力表示更新()
確認する
表示する()
ユーザーがボタン②を押すと「秒」を設定し「分」を点滅表示する
押す
通知する()
「分」設定開始()
タイマ更新()
時刻・カレンダ入力表示更新()
「秒」を設定()
現在の設定を確認する
表示する()
ボタン①を押すと，点滅中の数字が1進む
押す
通知する()
表示中の「分」を1進める()
タイマ更新()
時刻・カレンダ入力表示更新()
確認する
表示する()
ユーザーがボタン②を押すと「分」を設定し「時」を点滅表示する
押す
通知する()
「時」設定開始()
タイマ更新()
時刻・カレンダ入力表示更新()
「分」を設定()
現在の設定を確認する
表示する()
ボタン①を押すと，点滅中の数字が1進む
押す
通知する()
表示中の「時」を1進める()
タイマ更新()
時刻・カレンダ入力表示更新()
確認する
表示する()
ユーザーがボタン②を押すと「時」を設定し「日」を点滅表示する
押す
通知する()
「日」設定開始()
タイマ更新()
時刻・カレンダ入力表示更新()
「時」を設定()
現在の設定を確認する
表示する()
ボタン①を押すと，点滅中の数字が1進む
押す
通知する()
表示中の「日」を1進める()
タイマ更新()
時刻・カレンダ入力表示更新()
確認する
表示する()

〔図7〕分析シーケンス図

〔図7〕分析シーケンス図(つづき)

〔図8〕ブロック図

が明確になっているほうが，どのように実現するかという実装手段の選択の幅が限定されて実装しやすくなる．その制約の中で，システムができることを実現していくのが設計モデルである．分析モデルから設計モデルを作成することで，要求を実現するうえでのハードウェアとの整合がとれ，実際に動くシステムになる．したがって組み込みシステムを開発するにあたっては，分析モデルと設計モデルを分けて，実装までの開発作業を行うことが望ましい．

分析の作業と設計の作業についてのキーポイントは，

- 分析の段階では，設計や実装の制約なしに要求を実現するためのシステムを調べる
- 設計の段階では，分析で明確になった組み込みシステムに対する要求を実現するための制約条件を分析モデルに追加する

ということになる．

2.3 分析から設計への変換作業

ここでは，分析モデルから設計モデルに工程を移行する中で，必要な作業内容を解説する．

分析モデルから設計モデルを作成するにあたり，追加されるべき制約条件をどうやって洗い出したらよいだろうか．電波時計システムの開発では，ハードウェアでできることとソフトウェアで制御することの切り分けを行うために，ブロック図およびデバイス仕様書を作成した(**図8**，**図9**)．

ブロック図とデバイス仕様書からハードウェアの仕様を確認して，どのように制御しなければいけないかという制約を明らかにする．ハードウェアの仕様を確認することでソフトウェアの責務とハードウェアの責務の境界を明確にすることができる．

分析モデルにこれらの制約条件を加えて，現実のシステムと

〔図9〕デバイス仕様書

LCD Controller

モードレジスタ：設定値によって表示モードを切り替える
00：現在時刻表示モード(通常モード)
このモードでは，TMRにある現在時刻情報を直接Readして表示する．
カレンダについても直接Calendarの情報をReadして表示する．
01：RECEIVE表示モード(標準時刻電波受信表示モード)
02：時刻カレンダ設定モード
03：電波受信エラー日数表示モード
04：アラーム時刻表示モード
10：表示消去

RECEIVEモードモニタ表示レジスタ：受信モード時の表示を制御する
00：通常表示
01：RECEIVE表示
1F：ERROR表示（ドット表示部）
10：RCVD

報時時刻カレンダ設定モードレジスタ：モードレジスタが02の時有効
00：「報時＝OFF」を表示
01：「報時＝ON」を表示
11：現在時刻を表示しながら，「秒」表示を点滅表示
12：現在時刻を表示しながら，「分」表示を点滅表示
13：現在時刻を表示しながら，「時」表示を点滅表示
21：「日」表示を点滅表示
22：「月」表示を点滅表示
23：「年」表示を点滅表示
29：曜日表示部に「ERROR」を表示

アラーム設定時刻モードレジスタ：モードレジスタが04の時有効
00：通常表示
12：アラーム時刻を表示しながら，「分」表示を点滅表示
13：アラーム時刻を表示しながら，「時」表示を点滅表示

※マーク表示レジスタ
10：アラームマークを表示しない
11：アラームマークを表示する

TMR

Current Time Register：表示時刻データレジスタ
Hour Data Register　：現在の「時」Data　R/W可能
Minute Data Register　：現在の「分」Data　R/W可能
※00になるとInt-05にて割り込みを発生させる
Second Data Register　：現在の「秒」Data　R/W可能

Alarm　Time　Register：アラーム時刻設定レジスタ
Alarm Hour Data Register　：アラーム設定時刻の「時」Data　R/W可能
Alarm Minute Data Register：アラーム設定時刻の「分」Data　R/W可能
Alarm Interrupt Enable Register：アラームInt許可　R/W可能

Receive Time Data Register：受信時刻データレジスタ
Receive Hour Data Register　：受信した「時」Data　R/W可能
Receive Minute Data Register　：受信した「分」Data　R/W可能
Receive Second Data Register　：受信した「秒」Data　R/W可能

Calender

Current Date Register：表示暦データレジスタ
Date Data Register　：現在の「日」Data　R/W可能
Month Data Register　：現在の「月」Data　R/W可能
Year Data Register　：現在の「年」Data　R/W可能

デコーダ

デコーダは受信器で受信したシリアル情報を適切に抽出，分析し，年月日時分秒データに変換する．
デコーダは受信データのデコードが終了すると，割り込み信号(Int-04)を発生する．
ソフトは，割り込み発生後に受信データレジスタの内容を読み出すことで，電波標準時間を知ることができる

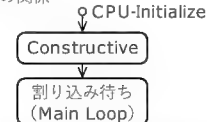
受信器

受信器は通常は電力消費を抑えるために，Out-03を0にすることで回路電源をOFFしておくこと．
受信器は電源が入れられると，受信したデータをそのままデコーダへ送る

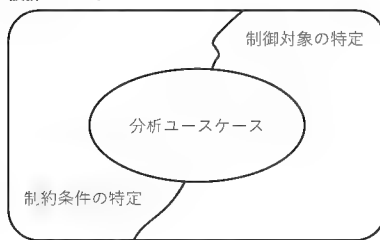
ブザー

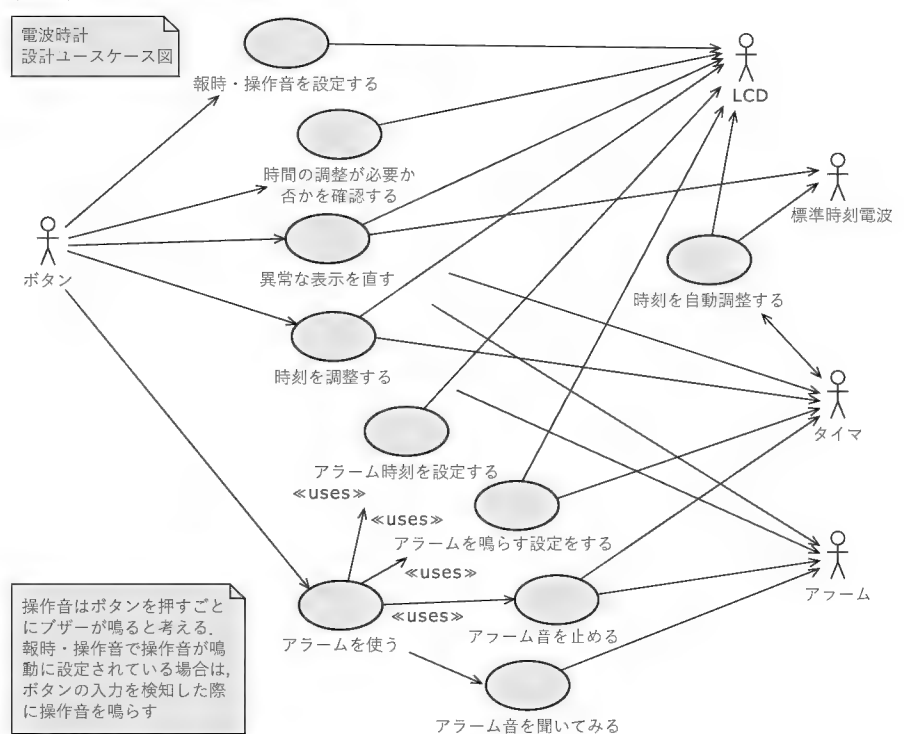
Out-01を1にするとブザー1は音を鳴らす．0にすると音を止める
Out-02を1にするとブザー2は音を鳴らす．0にすると音を止める

― MEMO ―
Monitor→割り込みハンドラとAPSの関係

CPU-Initialize
↓
Constructive
↓
割り込み待ち
(Main Loop)

- 割り込みベクタテーブルには直接Methodの絶対番地が書かれている
→割り込みが入ると直接Methodを呼び出し可能

〔図10〕分析ユースケースと設計ユースケースの関係

〔図11〕設計ユースケース図

して実現できるように詳細化することが設計モデルを作っていくことになる．

まず，設計ユースケースの作成では，ブロック図，デバイス仕様書から明らかになったソフトウェアシステム外の制御対象と，各分析ユースケースの関係を明確にすることから始める．特別な制御が必要な制御対象があれば分析のユースケースに対して設計ユースケースを増やしてもよいが，電波時計システムにおいては分析ユースケースと同じユースケースのまま制御対象を副ア

〔図12〕設計ユースケースシナリオ

Use Case Document Form 2.0

設計

版No.	発行日	作成者	承認
K1		石田栄子	

設計ユースケース：時刻を調整する

○**システム**

・電波修正時計システム

○**アクタ**

・メインアクタ：ボタン
・サブアクタ：デコーダ，タイマ，LCD

○**目的**

・ユーザーがシステムの時刻を調整する

○**概要**

・システムはボタン操作により標準時刻電波受信による時刻調整を強制的に行うことができる．正常に電波受信できない場合は時刻・カレンダ設定モードにて各項目を入力設定し，時刻を調整する

○**事前条件**

・なし

○**事後条件**

・システムは時刻の調整が完了し時刻表示モードになっている

○**シナリオ記述**

基本系列

シナリオ1．標準時刻電波の強制受信を行う場合
ユーザーは時刻表示モードであることを確認したうえで，ボタン②を2秒以上押し，システムに標準時刻電波の強制受信を指示する

アクタアクション	システムレスポンス
1. ボタンは押されたことを，Int-02でシステムに通知する	2. システムは，In-02が1であることを確認し，タイマに2000ms後に割り込みを発生するようにセットする
3. タイマは2000ms経過したことをシステムにInt-05で通知する	4. システムは，Int-02の解除割り込みがなく，In-02が1であることを確認し，LCDのモードレジスタに1を書き込み，ディスプレイに「RECEIVE」マークを点滅表示する
	5. システムはOut-03を1に設定し，受信器をONする
	6. システムはデコーダに標準時刻電波受信開始を設定する
7. デコーダは標準電波を受信し，システムにInt-04で通知する	8. システムは，デコーダの電波受信強度レジスタのデータを読み出し，良好に受信できていることを確認し，LCDのカレンダ表示部レジスタに[受信状況のドット表示]をするためのデータを書き込む
	9. システムは，デコーダの受信時刻レジスタのデータを読み出し，データが規定の範囲に収まっていることを確認する
	10. システムはドット表示部に「RCVD」が表示されるようにLCDの該当レジスタにキャラクタコードを書き込む
	11. システムは，Out-01を1に設定し，受信成功音を鳴らす．500ms後に割り込みをかけるようにタイマをセットする
	12. システムは，読み出した受信データを受信時間表示用キャラクタコードに変換し，LCDの受信時間表示レジスタに書き込む
	13. システムは，Out-03を0にすることで，受信器をOffする
	14. システムは，デコーダに受信処理終了ステートを書き込む
	15. システムは現時刻タイマを停止し，受信した時間データ＋処理時間のデータを設定する
	16. システムは現時刻タイマを起動する
17. タイマは500ms経過したことをシステムにInt-05で通知する	18. システムは，Out-01を0に設定し，受信成功音を止める
	19. システムは，5000ms後に割り込みをかけるようにタイマをセットする
20. タイマは5000ms経過したことをシステムにInt-05で通知する	21. システムは，調整モード表示から，時刻表示モードにするためにLCDのモードレジスタに0を書き込む

シナリオ2-1．ボタン操作により時刻調整項目の設定を行う場合

アクタアクション	システムレスポンス
1. ボタンは押されたことを，Int-03でシステムに通知する	2. システムは，In-03が1であることを確認し，タイマに2000ms後に割り込みを発生するようにセットする
3. タイマは2000ms経過したことをシステムにInt-05で通知する	4. システムは，Int-03の解除割り込みがなく，In-03が1であることを確認し，LCDのモードレジスタに02を書き込み，ディスプレイは[時刻・カレンダ設定モード]になり「報時」の選択状態を表示する
5. ボタンは押されたことを，Int-01でシステムに通知する	6. システムは，「報時_ON情報をトグル(排他的論理和)させ，その結果をLCDの報時時刻カレンダ設定モードレジスタに書き込む(Int-01が起こるたびにこの動作を繰り返す)
7. ボタンは押されたことを，Int-02でシステムに通知する	8. システムは6の結果を制御データとして記憶する
	9. システムは，LCDの報時時刻カレンダ設定モードレジスタに11を書き込み，時刻・カレンダを現在の設定で表示し「秒」に00を点滅表示する
10. ボタンは押されたことを，Int-01でシステムに通知する	11. システムは「秒」を＋1する(Int-01が入るごとに＋1する)
12. ボタンは押されたことを，Int-02でシステムに通知する	13. システムは現在の「秒」データをタイマのカレントタイムレジスタのSecond Data Registerに書き込む
	14. システムは，LCDの報時時刻カレンダ設定モードレジスタに12を書き込み，タイマのMinute Data Registerのデータを「分」データとして点滅表示する
15. ボタンは押されたことを，Int-01でシステムに通知する	16. システムは「分」を＋1する(Int-01が入るごとに＋1する)
17. ボタンは押されたことを，Int-02でシステムに通知する	18. システムは現在の「分」データをタイマのカレントタイムレジスタのMinute Data Registerに書き込む
	19. システムは，LCDの報時時刻カレンダ設定モードレジスタに13を書き込み，タイマのHour Data Registerのデータを「時」データとして点滅表示する
20. ボタンは押されたことを，Int-01でシステムに通知する	21. システムは「時」を＋1する(Int-01が入るごとに＋1する)
22. ボタンは押されたことを，Int-02でシステムに通知する	23. システムは現在の「時」データをタイマのカレントタイムレジスタのHour Data Registerに書き込む
	24. システムは，LCDの報時時刻カレンダ設定モードレジスタに21を書き込み，CalendarのDate Data Registerのデータを「日」データとして点滅表示する
25. ボタンは押されたことを，Int-01でシステムに通知する	26. システムは「日」を＋1する(Int-01が入るごとに＋1する)
27. ボタンは押されたことを，Int-02でシステムに通知する	28. システムは現在の「日」データをカレンダのカレントデートレジスタのDate Data Registerに書き込む
	29. システムは，LCDの報時時刻カレンダ設定モードレジスタに22を書き込み，CalendarのMonth Data Registerのデータを「月」データとして点滅表示する
30. ボタンは押されたことを，Int-01でシステムに通知する	31. システムは「月」を＋1する(Int-01が入るごとに＋1する)
32. ボタンは押されたことを，Int-02でシステムに通知する	33. システムは現在の「月」データをカレンダのカレントデートレジスタのMonth Data Registerに書き込む

〔図12〕設計ユースケースシナリオ（つづき）

	34. システムは，LCDの報時時刻カレンダ設定モードレジスタに23を書き込み，CalendarのYear Data Registerのデータを「年」データとして点滅表示する
35. ボタンは押されたことを，Int-01でシステムに通知する	36. システムは「年」を＋1する（Int-01が入るごとに＋1する）
37. ボタンは押されたことを，Int-02でシステムに通知する	38. システムは現在の「年」データをカレンダのカレントデートレジスタのYear Data Registerに書き込む
※ボタン1が押された場合は，5.～38. を繰り返す	
39. ボタンは押されたことを，Int-03でシステムに通知する	40. システムはLCDのモードレジスタに00を設定し，時刻表示モードにする

シナリオ2-2．時刻調整項目の設定中にユーザーが操作を止めた場合．[シナリオ2-1. の2.以降のどこでも]

アクタアクション	システムレスポンス
1. ボタンは押されたことを，Int-01，Int-02，Int-03でシステムに通知する	2. システムは，タイマに120秒後に割り込みを発生するように設定する
3. タイマは120秒経過したことをInt-05で通知する	4. システムはLCDの報時時刻カレンダ設定モードレジスタの値を確認し，対応する設定データを該当するデータレジスタに設定する
	5. システムはLCDのモードレジスタに00を設定し，時刻表示モードにする

代替系列

シナリオ１の代替系列1．標準時刻電波の強制受信を行い，正常に受信ができない場合

アクタアクション	システムレスポンス
1. ボタンは押されたことを，Int-02でシステムに通知する	2. システムは，In-02が1であることを確認し，タイマに2000ms後に割り込みを発生するようにセットする
3. タイマは2000ms経過したことをInt-05で通知する	4. システムは，Int-02の解除割り込みがなく，In-02が1であることを確認し，LCDのモードレジスタに1を書き込み，ディスプレイに「RECEIVE」マークを点滅表示する
	5. システムはOut-03を1に設定し，受信器をONにする
	6. システムはデコーダに標準時刻電波受信開始を設定する
7. デコーダは標準電波を受信し，システムにInt-04で通知する	8. システムは，デコーダの電波受信強度レジスタのデータを読み出し，良好に受信できていることを確認し，LCDのカレンダ表示部レジスタに[受信状況のドット表示]をするためのデータを書き込む
	9. システムは，デコーダの受信時刻レジスタのデータを読み出し，データが規定の範囲にないことを確認し，LCDのReceiveモードモニタ表示レジスタに02を書き込みERRORを表示する
	10. システムはOut-01を1にして，Error音を鳴らし，タイマに300ms後に割り込みを発生するように設定する
11. タイマは300ms経過したことをInt-05で通知する	12. システムはOut-01を0にして，Error音を止める．システムはタイマに5000ms経過後に割り込みを発生するように設定する
13. タイマは5000ms経過したことをInt-05で通知する	14. システムは，LCDのモードレジスタを00にし，現在時刻表示モードに設定する．また，報時時刻カレンダ設定モードレジスタに29を書き込み，曜日表示部にERRORを表示する

シナリオ1の代替系列2．受信動作をユーザーが中断し時刻の調整ができない場合

アクタアクション	システムレスポンス
1. ボタンは押されたことを，Int-02でシステムに通知する	2. システムは，In-02が1であることを確認し，タイマに2000ms後に割り込みを発生するようにセットする
3. タイマは2000ms経過したことをシステムにInt-05で通知する	4. システムは，Int-02の解除割り込みがなく，In-02が1であることを確認し，LCDのモードレジスタに1を書き込み，ディスプレイに「RECEIVE」マークを点滅表示する
	5. システムはOut-03を1に設定し，受信器をONにする
	6. システムはデコーダに標準時刻電波受信開始を設定する
7. ボタンはInt-01，02，03のいずれかが発生したことを通知する	8. システムは，Out-03を0にすることで，受信器をOffにする
	9. システムは，デコーダに受信処理終了ステートを書き込む
	10. システムは，LCDのモードレジスタに00を書き込み，現在時刻表示モードを設定する

○**注意点**

・ユーザー操作で項目設定する場合は，秒分時日月年はそれぞれボタン①が押されるたびに表示中の数字から一つ進む．秒・分は59の次は0，時は23の次は0，日は31の次は1，月は12の次は1，年は2045の次は1996となる．ボタン③を押すまで報時→秒→分→時→日→月→年の設定を繰り返す

○**Note**

・曜日は，年月日よりシステム内で算出するため入力設定はできない
・以下のアクタからのイベントが発生しなかった場合，システムは致命的に故障しているのでソフトでは対処しない
　・基本系列1-3　タイマの2000ms経過割り込みが入らない場合
　・基本系列1-7　デコーダからの通知がなかった場合
　・基本系列1-17　タイマの500ms経過割り込みが入らない場合
　・基本系列1-20　タイマの5000ms経過割り込みが入らない場合
　・基本系列2-1-3　タイマの2000ms経過割り込みが入らない場合
　・基本系列2-1-5　Int-01割り込みが入らない場合
　・基本系列2-1-7　Int-02割り込みが入らない場合
　・基本系列2-1-39 Int-03割り込みが入らない場合

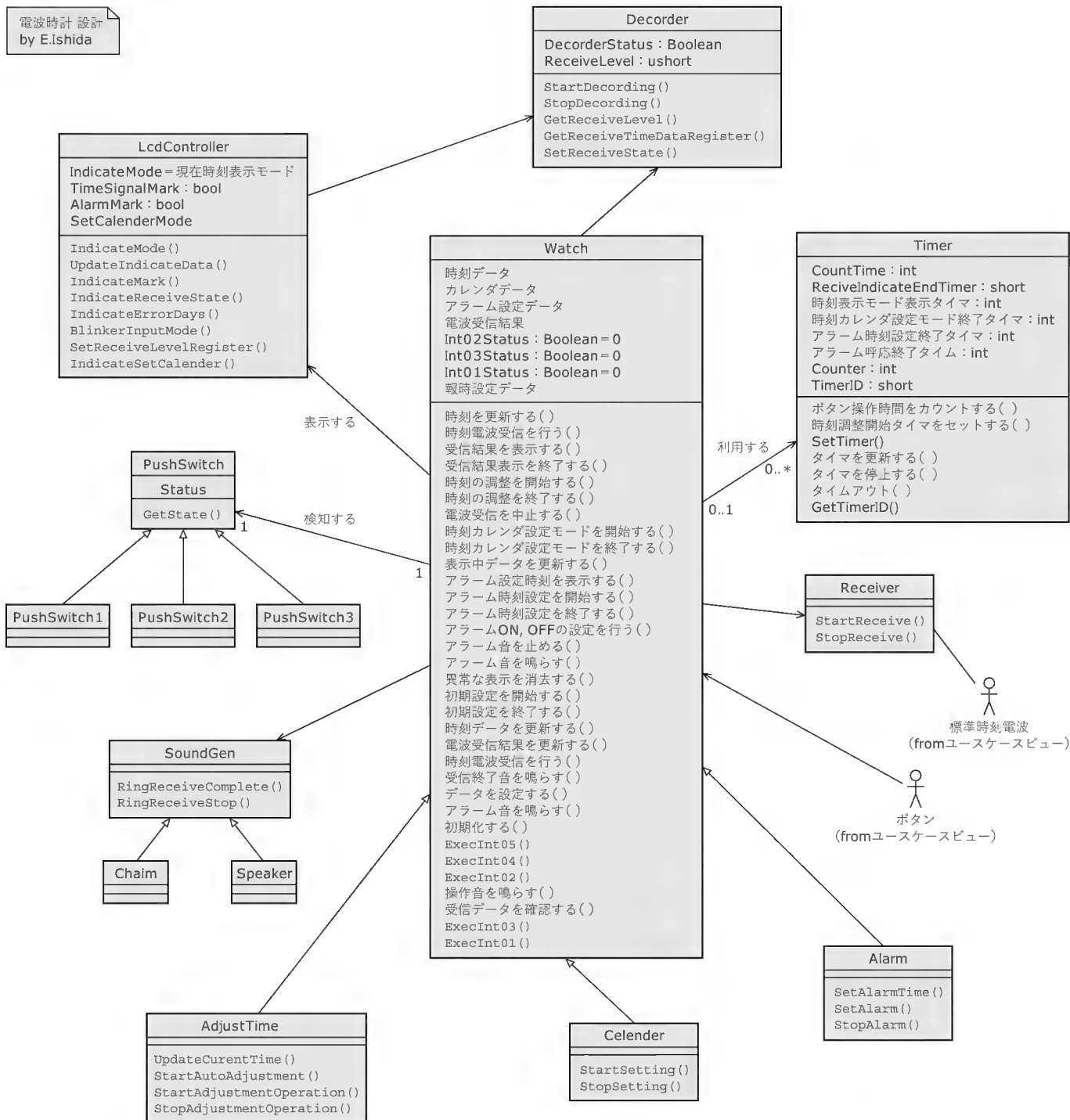
改訂NO.	変更者	変更日	変更箇所	変更内容	備　考

クタとして追加し，分析で抽出したユースケースとの関連を表した（図10，図11，p.83）．

基本的に設計ユースケースシナリオは，分析ユースケースシナリオに「どのように」実現するかの手順を加えていく．ブロック図，デバイス仕様書から，ハードの使い方がわかるので，デバイスそれぞれを動作させるためのプログラムが実装できるレベルまで具体的にシナリオを書き込んでいく（図12）．

設計クラス図は分析クラス図を元に作成するが，ハードウェアの仕様を明確にしたことで新たに追加された実装のためのデバイスをコンポーネントクラスとして追加するとともに，実装のために必要な概念クラス，具体的にはモニタや通信ドライバといったクラスを抽出し，詳細化したクラス図と関係付けを行う．クラスの追加を行いながら設計ユースケースのシナリオに合わせてクラス図のクラスをトレースし，設計ユースケースを実現するための各クラスの操作を追加し，クラス定義書，セマンティクス記述（操作定義書）の作成に結び付けていく（図13，図14）．

実装については，設計で作成したクラス図の詳細化で実現できるので，あえて説明する必要はないだろう．設計クラス図のクラス定義書・操作仕様書を元に，クラスごとに属性，操作を

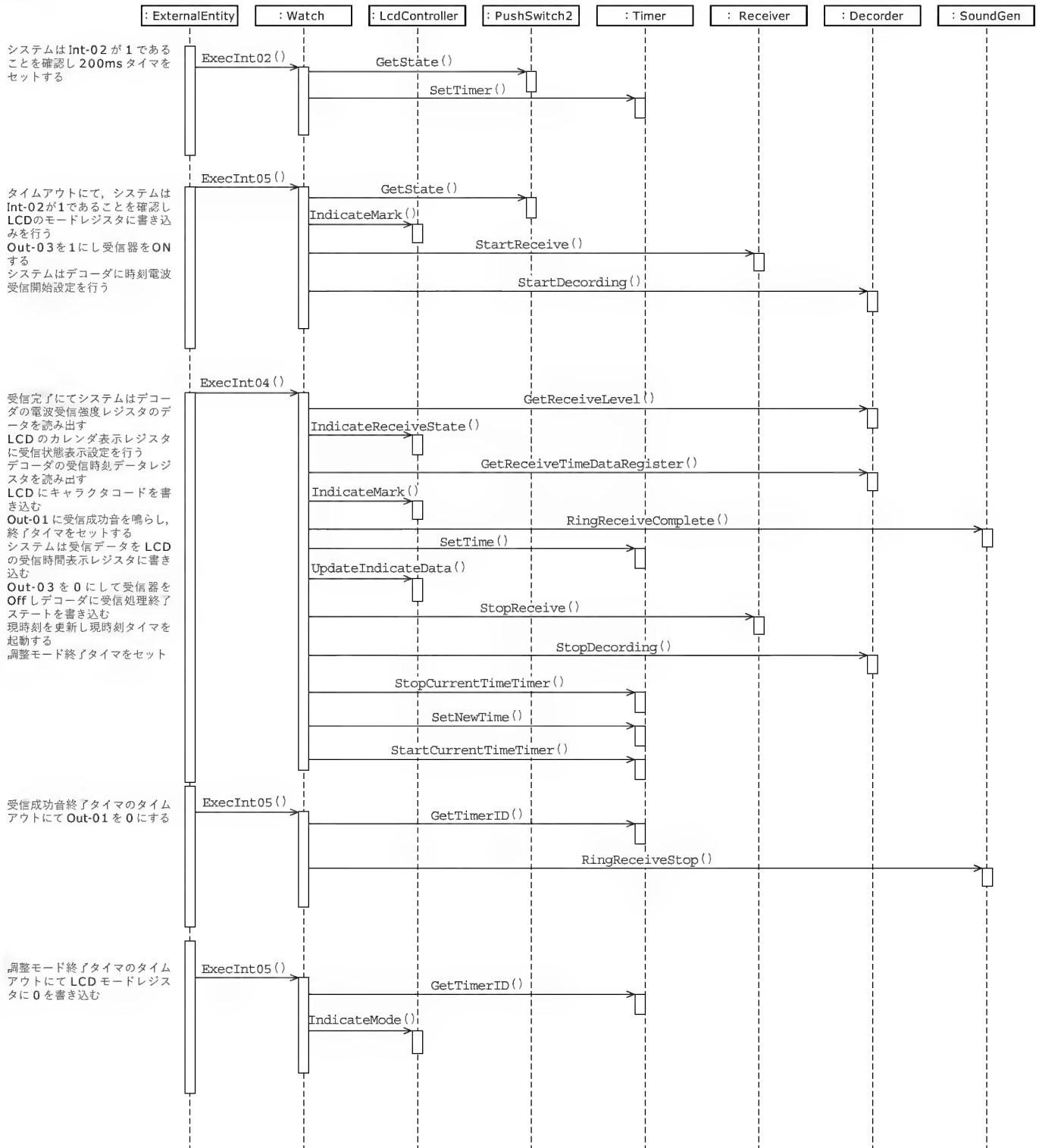
〔図13〕設計クラス図

実装していけば，ソースファイルが作成できる．

3 検証と革新の方向性

3.1 分析と設計が一致することによるアドバンテージ

分析モデルの構成をそのまま利用したモデルによる設計・実装を進めることで，直感的に理解しやすい構成を意識する必要がある．

理想的な分析のモデルは普遍的なモデルで，直感的に把握しやすいモデルである．設計のモデルは分析の結果を実現するための方法なので，分析のモデルの普遍性に実現手段を加えてしまう．このため，分析モデルの普遍性を維持できなくなる．分析のモデルを元にその普遍性を生かしたままの設計モデルにすることで，現在だけでなく何年か先に誰が見ても納得できる設

〔図14〕設計シーケンス図

New Products ― モンタビスタとソリッド，組み込み用分散型データプラットホームを提供
モンタビスタとSolid Information Technologyは，MontaVista Linux上で，分散型データ管理プラットホームSolid FlowEngineの動作検証を行い，完全対応したことを発表した．

〔**表 2**〕**分析ユースケースと設計ユースケースの比較**

分析ユースケース名	分析から設計で変わったこと	変わった理由
時刻を知る	●削除した	●ソフトで実現する必要がない
時刻の調整が必要か否かを確認する	●アクタをボタンにする ●サブアクタを追加	●ユーザーとシステムのインターフェースを明確にする ●システムが制御するものをサブアクタとして明確にする
時刻を調整する		
報時・操作音を設定する		
異常な表示を直す		
時刻を自動調整する		
アラームを使う	●アクタをボタンにする ●他のアラームに関するユースケースの抽象ユースケースとする	●ユーザーとシステムのインターフェースを明確にする ●抽象ユースケースとして他のアラームに関するユースケースと区別する
アラーム時刻を設定する	●「アラームを使う」ユースケースの使用/汎化ユースケースとする ●サブアクタを追加	●「アラームを使う」ユースケースに使用される関係を明確化
アラームを鳴らす設定をする		
アラーム音を止める		
アラーム音を聞いてみる		

〔**表 3**〕**分析クラス図と設計クラス図の比較**

分析クラス名	設計クラス名	分析から設計で変わったこと	変わった理由
ボタン	Push Switch	システムとの関係を変更	実現手段としてシステムがボタンの状態を検知することを明記
ボタン1	Push Switch1	なし	
ボタン2	Push Switch2	なし	
ボタン3	Push Switch3	なし	
ディスプレイ	LcdController	Decorderとの関係を追加	追加した制御対象との関係を明記
ウオッチ	Watch	システムとして制御対象との関係を追加・変更	実現手段を考えた関係に変更
タイマ	Timer	システムとの関係を変更	実現手段としてシステムがタイマを使用することを明記
	SoundGen	追加	実際の制御対象として登場させる
	Chaim	追加	実際の制御対象として登場させる
	Speaker	追加	実際の制御対象として登場させる
	AdjustTime	追加	時刻調整のふるまいを一つのクラスとして制御することを明確にする
	Calender	追加	日時設定のふるまいを一つのクラスとして制御することを明確にする
	Alarm	追加	実際の制御対象として登場させる
	Receiver	追加	実際の制御対象として登場させる
	Decorder	追加	実際の制御対象として登場させる

計モデルが提供できる．

分析モデルをベースに設計モデルで制約条件を満たした設計を行うことで，システムができること，すなわち要求仕様をそのまま実装につなげることができるだろう．分析モデルの構成で設計・実装を進めることで，システムの目的を明確にしたまま実装を行える．システムの構成が理解しやすければ，分析モデルから一貫性をもった実装が可能になる．一貫性が保たれるということは，オブジェクト指向がめざす効果的な再利用が促進されるということを意味する（**表 2**，**表 3**）．

3.2　詳細化がモデルをわかりにくくする原因

設計から実装の各作業にともなう各制約条件は，直感的にわかりやすいソフトの静的構造をわかりにくい形に変えていく．なぜわかりにくくなるのか？　何がわかりにくくさせているのかについて，考察する．

現実の開発では，設計工程で作成したモデルと実装が直結していないことが多々ある．せっかく設計を進めてきても，コードが設計と違っていては意味がない．実際にコードを組みながら設計にない部分ができてしまうのはなぜだろう？

いわゆる実装テクニックのようなものが，分析結果から一貫性を意識して設計してきた結果を実装時に崩してしまっているのではないだろうか？　本当にしたいこと以外のコードがあると，何のためにそのコードがあるのか理解しにくくなる．

コーディング規約のような実装時のルールの取り決めが，設計モデルとソースコードのつながりをわかりにくくすることもある．コードを見やすくしようとしたり，ファイル管理をしやすくすることを優先することで，本来進めるべき設計モデルの詳細化とは違うコードができてしまう．さらに，コーディング規約が守られていないこともあり，こうなってしまうと設計と実装のつながりは，ますますいいかげんな状況になる．

コーディング規約を守らない原因としては，開発担当者がこの規約を理不尽な規約と感じる場合があることで，組織として強制力をもったルールとして位置付けないかぎりはしたがう気がしなくなり，実際にしたがわないといった状況が起こる．また，コーディング規約のメンテナンスも難しく，うまく運用しないと規約が変わるたびにその内容について再確認をするための時間をかけることになる（**表 4**）．

そもそも実装言語自体に言語仕様があるのだから，その制約以上に新たに制約を作る必要があるのだろうか？　本来は，コ

〔表4〕実装時に設計モデルをわかりにくくする要因

わかりにくくする要因の事例	説　明
同じような意味のデファインがいくつもある	OK，NGとTrue，Falseなどが統一できていない
RAM領域に宣言された変数をデファイン宣言	デファイン名で操作していたり変数のまま操作していたりして，実際にどこでデータが変化しているかわかりづらい
コーディング規約があるのにそれを守りきれない	コーディングスタイル自体が細かすぎたり，規約が多すぎたりして，担当者が把握できない．忙しさを理由に規約を無視して突き進んでしまう．コーディングスタイルによっては現場の負荷になる
ヘッダファイルとソースファイルの切り分けが曖昧	それぞれをどのように構成するかについての規定があいまいなコーディング規約の場合がある．ヘッダの有無や，定義個所の分散などの定義が標準的でないことがあり，新規に開発する場合には現場開発者が混乱することがある
ヘッダファイルと共通ヘッダファイルの切り分けが曖昧	それぞれの役割分担が不明確なまま双方をメンテナンスする必要にせまられる
メソッドの命名	同じような意味の名前がいくつもあると，どれをどの場合に使ったらよいのかわかりづらくなる
デファインスイッチによる実行ファイルの制御	実行ファイルの切り替え制御を設計資料から読み取ることができない．別の資料が必要だったり，ソースコードを見てはじめて切り替えをしていることを知ったりする

ンパイラなどの言語処理系のツールでコーディングスタイルのチェックをするべきではないだろうか？　それ以上に決め事をするのは実装の制約事項を増やしていることになり，開発効率を改善するための活動としては本末転倒である．

3.3　モデルの読み方について

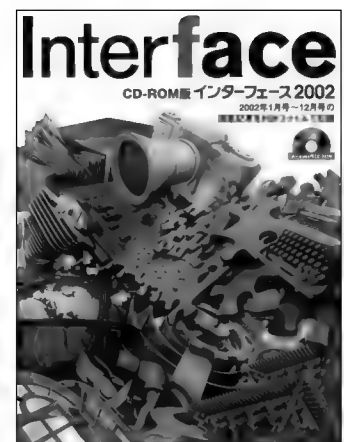
モデルを見る側に必要なスキルは何か，モデルを書く側に必要な考慮(配慮)は何かについて考察する．

モデルを書くときは，誰にでもわかりやすくという配慮が必要だ．モデルを読む人が何の目的でそのモデルを読むのかということまで考えたうえで，システムの目的と構成を正しく伝える必要がある．作成者がいなければわからないようなモデルや，作成者がモデル以外の説明手段で解説することではじめて理解できるようなモデルでは，オブジェクト指向が目的としている再利用を達成できているとはいえない．難解なモデルは作成者しか理解できないので，現在の組み込みシステムに要求されるような複数の開発者で協業して一つの製品の機能を実現するような開発には向かないといえる．

モデルを読む側に，UMLの基本ルールを知ることが最低でも必要になる．基本ルールを習得するためには，なるべく多くのモデルについて実装コードを含め見る機会をつくることで，このことがモデルを見る目を養うことになる．また，自分でモデルを書き，実装コードまで作成してみることが，モデルを解析するスキルを身につける近道である．

オブジェクト指向で分析，設計，実装を実際に行ったからといっても，開発作業が簡単になるとはいえない．従来の開発以上に検証を重ねるのがオブジェクト指向による開発と考えるべきである．確実に分析，設計を進めることでモデルと実装がかけ離れることなく，システムの目的と実装手段が同時に読み取れる良い設計ができるのではないだろうか[注1]．

いしだ・えいこ　(株)ケイ・エイチ・ジェーサービス
すぎうら・ひでき　富士ゼロックス(株)

注1：本章で解説した電波時計システムについて，実装コードを本号付属CD-ROM InterGiga No.30に収録している．参考にしていただきたい．

第4章

二つのアプローチを比較し，検討する

組み込みソフト開発にオブジェクト指向を導入した成果の測定手法

杉浦英樹

これまで「エレベータモデル」(第2章)，「電波時計モデル」(第3章)という，設計方針の異なるモデルの開発について解説してきたが，本章ではこの二つのモデルへのアプローチを比較しながら，それぞれのメリット/デメリットを説明する．そして，組み込みソフトの開発にオブジェクト指向を導入するに際し，成果をどのように測り，評価するかなどについて解説する．

(編集部)

1 組み込みソフトのモデルはわかりにくい？――検証と革新の方向性

1.1 わかりやすいモデリング？

分析モデルの構成をそのまま利用したモデルによる設計，実装を進めることで，直感的に理解しやすい構成をとることができる．これは，「何を」に着目した分析結果が，制御対象や制御そのものの普遍性をもっているためで，「どうやって」から発想する場合に比べてより上位の概念でモデルが構成されているからである．

「何を」，「どうやって」実施するかが設計だが，その施策は複数存在する．実装するシステムのもつ制約条件をすべて考慮して，システムの目的・方針にしたがって実装するわけだが，「何を」作るのかがわかっていれば，制約の中での最適解が見つけやすいわけである．したがって，設計者は常に分析の結果を意識してモデリングをする必要がある．実装に向けての各制約条件は，直感的にわかりやすいソフトの静的構造をわかりにくい形に変えていく．なぜわかりにくくなるのか？

前述したように，分析による実現内容の特定は，実現手段を考慮しなければしないほど普遍性をもつことになり，実現手段を考慮しないことで淘汰されたモデルは，多くのエンジニアが容易に理解できるようなものになる．ところが実装では，実装する手段としてのシステムのもつ独自の制約が存在するため，システムのもつ制約が理解できていなければモデルを理解できないことになる．

たとえば，モータを駆動する場合，ステッピングモータなのかDCモータなのか，電圧制御なのか電流制御なのか，回転数の制御方法はどうなのかという制約条件がシステムによって一義的に特定されるが，モータ駆動に関する基本的な知識や制御対象の特性に関する基本的なスキルがなければ，設計モデルから制御方法や制御対象を想像することは，ほとんど無理ということができる．

さらに，組み込み型ソフト開発の分野では，特化したハードウェアと独自の基本ソフト(モニタやリアルタイムOSなど)とデバイスドライバといわれるソフトとの関係を作って組み込みアプリケーションを構築するので，組み込みアプリケーションの設計モデルには，それらとの関係が示される必要がある．組み込みシステムはコスト上の制約や，スペース上の問題などから特化したハードウェアを制御することがほとんどなので，同じ組み込みソフト開発エンジニアでも，他の組み込みソフトの設計モデルを容易に理解できないことが多いのではないだろうか？

組み込みソフトエンジニアは，(理想的には)開発対象の周辺技術すべてに精通したプロフェッショナルなので，その問題領域については深い知見をもっているが，だからといって他のシステムを簡単に理解したり開発できるというわけではない．

では，どうすれば他のシステムを簡単に理解できるようになるのだろうか？

組み込みソフトエンジニアは，自分の開発対象について，ユニークな部分を自覚するべきであり，そのユニークな部分についての理解を促進するためのモデリングを心がけるべきである．ユニークな部分が開発対象の独自性や強みになっているはずなので，それに対する認識を深めるとともにさらに磨きをかけるなどの工夫ができるだろう(**図1**)．

組み込みシステムを構築する際に，制御ソフトの構成を階層的にモデル化することが行われるが，制御階層のレベルに応じた各モデル間の関係を明確にするだけでも設計モデルの理解度が促進できる．

1.2 ユースケースのシナリオ作成が大事

モデル作成者は，クラス図だけに傾注しがちだが，そのクラス図を理解するために必要なユースケースのシナリオ作成にもっと力を入れるべきである．シナリオが自然言語で，誰にでもわかりやすく書かれていれば，クラス図を理解するのに役立つわけである．

また，コンセプトペーパーによるシステム要求の整理や制約

条件の確認が有効なことは，先の電波時計の例でもあげたとおりである．

組み込みソフトをわかりにくくさせているのは，わからせることの価値に気付いていないか，あえてわからせることをしない一部のエンジニアによる不健全なモデルへの介入によるところもあるかもしれない．

知的付加価値を提供することで社会に貢献しているという自覚があれば，定式化された技術は公のものにし，新たな課題の解決に着手すべきであるという基本理念が生まれ，誰にでも理解しやすい設計モデルが描けるのかもしれない．

さて，ここで視点を変えて，モデルを見る側に必要なスキルとは何かを考察する．UMLは表記法を提供しているが，表記法だけを理解しても，そのモデルが何をしているかの理解にはつながらない．モデルを理解するためには，再び何をしようとしているのかに戻る必要があり，自然言語で記述するユースケースなくしてはモデルの理解は促進できないといえる．

もちろん，クラス図が多くを語るわけだが，そのクラス図が「何を」,「どうする」ためのクラス図なのかを解説しているのはユースケースのシナリオだけだといっても過言ではない．ユースケースとクラス図を考察する能力が読み手にあれば，モデルの理解は大幅に促進されるといえる(**図2**).

〔**図1**〕**組み込みエンジニアに必要なスキル領域**

〔**図2**〕**ユースケースシナリオとクラス図の関係の整理**

クラス図だけで，このシステムがやろうとしていることがわかるだろうか？
ユースケース図やユースケースシナリオが自然言語で語る説明が，クラスをもっともよく説明しているといえる

ユースケース図

シナリオ1：外気温>内気温のとき

アクタアクション	システムレスポンス
1．アクタはシステムを起動する	2．システムは温度センサで外気温>内気温を確認し，(吸気する)CW方向にMotorを回転する
	3．システムは温度センサで外気温＝内気温を確認し，Motorを停止する

ユースケースシナリオ

2 エレベータモデルと電波時計モデルの違い

2.1 Model Driven Architectureに近い考え方 ──「エレベータモデル」

第3章，第4章で作成した二つのモデルは，それぞれ大きく設計方針が異なるということはお気付きのとおりである．

エレベータモデルでは，クラス図から実装言語の接続が直感的にわかりづらいが，たとえばコード自動生成ツールを利用するような開発で，実装言語をまったく気にせずに開発できるようになれば，モデル(クラス図や状態遷移図)の形，関連や遷移の仕方そのものが制御内容を示すということができる．再利用や拡張性，柔軟性に富んだ組み込み制御システムをクラス図で提供することができそうである．この方向性は，最近話題になってきたMDA(Model Driven Architecture)に近い考え方といえる．

モデルでソフトを考える利点としては，たとえば，一般にOMTなどの方法論では，実装時の確認や検査のしやすさ，問題発生時の追跡のしやすさ，継承元の管理の難しさから多重継承を避け，どうしても多重継承を利用する場合には，とくに継承内容を重点管理することで対処している．このことから，多重継承は再利用開発には向かないように考えられる．

ところが，今回のエレベータモデルのように，モデルそのものが実装を表し，実装言語による制限や管理上の制約はコード自動生成ツールの機能を利用してチェックできるのであれば，本当の意味で実装の制約から開放される．こうなると多重継承の難しさやわずらわしさを払拭できるので，多重継承の利用にともなう不具合の可能性を気にしたり，追跡時にわかりやすいように重点的に管理する必要がなくなる．ここでいいたいのは，実装技術が進歩することで，ソフトの開発方法が進化するということである．

ただし，この方法はモデリングの技術の敷居が高く，実装とのつながりで議論できない分，裾野が狭いということもできる．つまり，ソフトウェア開発の工業化という意味では，誰にでもできるような技術にするには方法論の確立や教育の充実をはかる必要がある．現在の開発環境や教育環境のままでは，開発環境の面からは一部のリッチな企業でしか採用できないような高価なツールを準備する必要があったり，教育の面からは一部の

特殊なスキルをもったエンジニアしか使えないといった手工業的な状況が続きそうである．

2.2 ユースケース駆動型のアプローチ ――「電波時計モデル」

一方，電波時計モデルでは，すでに確立されているユースケース駆動型のアプローチで，かつ，分析と設計の一貫性を追及した感覚的にとらえやすいモデルを作っている．この方法では，実装言語までモデルの一貫性が確保できるので，どの工程の作業からでも追跡できることになる．つまり，開発者は，自分の得意なところからモデル全体を追跡することで，学習しながら知見を深めることができるのである．

エレベータモデルの例と比較して，開発者からみると裾野が広いことは間違いなく，工業化された方法といえる．再利用する担当者はその一貫性からシステムを把握しやすく，再利用時には変更点の特定や拡張ポイントの設定が容易に行えるはずだ．

再利用に関しても，差分で開発を進めることができれば，分析から検査までの作業を最小限に抑えることができるので，ソース自動生成ツールで開発する場合に比べても大差ない開発効率が得られるはずである．

2.3 どちらを使うか？

どちらの方法を選択するかは，開発環境の状況や開発対象との相性，開発担当者の好みによって変わるべきである．エンジニアサイドから見た場合には，開発方法の選択肢を増やすことでユニークな発想やオリジナルな開発方法を実践できる．たとえば，今回示した二つの例の良いところを組み合わせ，まったく新しい方法で開発することも可能である．

組み込みソフトの現在の状況を見ると，海外(とくに欧米)からの開発環境の輸入に加え，開発方法論やソフトウェア開発管理についての方法もほとんどが海外に端を発しており，日本の組み込みソフトエンジニアの一人としては非常に危機感を覚える．

海外に学ぶのは間違いではないが，日本の優位性として期待されている組み込みシステムの開発現場において，開発手法まで海外に依存してよいのだろうか．組み込みソフトエンジニアはもっと真摯に問題にあたり，ユニークな開発技術を創出し，今とはまったく逆に，日本の組み込みソフト開発技術を海外に輸出しても良いのではないだろうか．

いずれにせよ複数の方法を実践し，経験することで組み込みソフトエンジニアの「ソフト開発技術に対する興味」を増すことができれば，一歩あゆみをすすめることになる．

〔図3〕投資回収のシナリオ/損益分岐点

3 投資と回収をどう計るか

イノベーションとしてのオブジェクト指向を評価する意味で，投資回収効果の算定は避けて通れない．ここでは，投資と回収をどのように考えるべきかを考察する．ソフト開発の投資としてとらえるべき内容は，人件費，教育費，開発環境費の3項目で，それぞれに対する回収のストーリーが成立しなければ，ビジネスとしては成立しない．

最初のステップは，従来の開発からパラダイムをシフトさせるわけだから，従来のパラダイムではかからない再投資部分に着目しなければならない．つまり，旧パラダイムから新パラダイムへ移行する際に必要な投資内容のすべてを網羅し，それらの投資をしてもパラダイムシフトが有効であり，再投資した分を回収できるというストーリーがなければならないわけである．

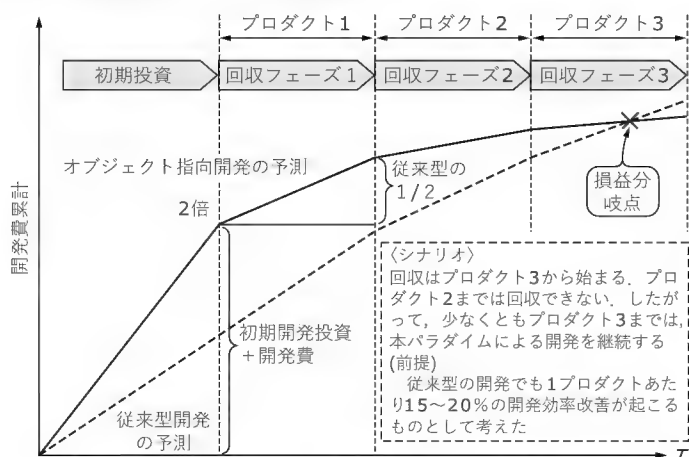
具体的には，旧パラダイムのソフト開発生産性を正しく把握したうえで，その開発生産性に対して，パラダイムシフト後の開発生産性がどれだけ効率化できるかということを予測する必要がある．その結果，投資に対する回収のポイントが見えてくるわけである．

横軸を時間，縦軸を回収金額とした場合に，旧パラダイムの開発効率の改善率に対して，新パラダイムの改善率が明らかにそれを上回る必要がある．また，回収金額の絶対値についても，新パラダイムのほうが上回り，旧パラダイムと比較することにより損益分岐のポイントが見えてくるわけである(**図3**)．

このように損益分岐点は，旧パラダイムで開発した場合の開発費累積と新パラダイムで開発した場合の開発費累積がクロスするポイントで知ることができる．したがって，旧パラダイムでの開発費の予測と，新パラダイムでの開発費の予測を公平に行う必要がある．

また，採用した技術の効果を投資効果として判断する場合には，その技術の効果が計れるように工夫する必要がある．とくに新パラダイムのメトリックスを選定する際には，旧パラダイムでの生産性指標と比較するための項目を忘れないように注意すべきである．

4 初期開発の測定

オブジェクト指向開発へパラダイムシフトする際に必要な投資に関しては，先に述べた，①人件費，②教育費と③開発環境

費を切り分けて測定する必要がある．

実際の初期開発に必要な投資は，

① 人件費としては，新規開発にともなう開発工数増加分，新パラダイムにおける新管理方法への対応としての管理工数の増加分

② 教育費に関しては，基礎教育，開発環境の教育として，社外講習会やコンサルタント，メンターなどを利用することを考える．また，旧パラダイムで計上していたドメイン知識習得のための通常教育（社内セミナや自主的な勉強会など）の工数は，旧パラダイムとの比較を厳密に行うためにすべて計上する

③ 開発環境費に関しては，初期設備設置費用として，PCやWSの設置，ネットワークインフラなどの固定設備費と開発に必要なCASEツールなどのソフト資産費を明らかにする．ここでも旧パラダイムとの比較を厳密に行う必要があるので，開発に必要なものはすべてもれなくリストアップすべきである．忘れがちなのが運用にかかるコストだが，保守費，メンテナンス費なども必要経費として計上する

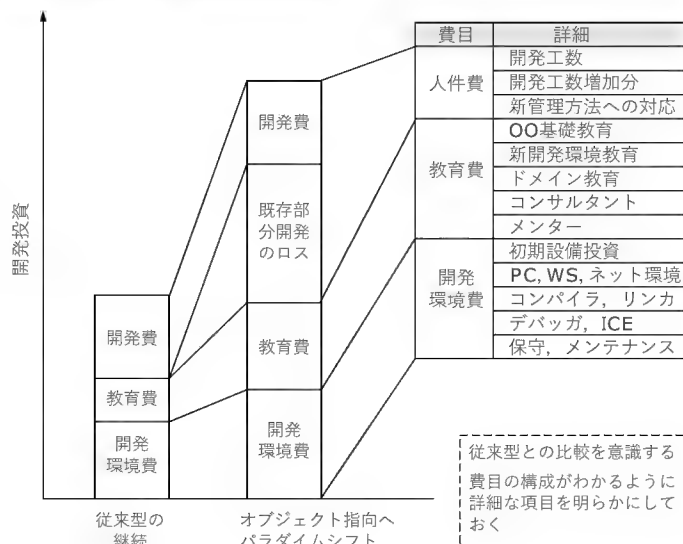
パラダイムシフト時は教育費が増加することに加え，旧パラダイムで継続していた流用・再利用がなくなる．その部分を新パラダイムで作り直すので，従来の開発ではやらなくてもよかった作業が起こり，一時的には旧パラダイムに比べて開発効率が落ちることになる．パラダイムシフトが完了すれば，旧パラダイムよりも開発効率が改善されるので，中長期的に見たときに，旧パラダイムで実施した場合に比べて開発効率が改善されていることが見えるように測定すべきである（**図4**）．

〔**図4**〕**投資内容と開発効率の測定**

5 実プロダクトに対して投資する際の注意

初期開発時には従来に対して大きな開発投資をするわけだが，この追加投資がパラダイムシフトに使用されているか否かが重要な関心事であり，使用されていないとなると重大な問題になる．従来のパラダイムで開発し，開発に対する投資が増えれば当然，そのプロダクトは「いままでより早く」できたり，「いままでより品質が高く」なるわけだが，旧パラダイムによる開発をしていても，投資が増えれば適応したプロジェクトで「いままでより良くなる」のはあたりまえで，投資に対する即効性，すなわち即座に回収できる場合には，まずパラダイムシフトしていないと考えるべきである．

問題になるのは，その次や次の次のプロジェクトでオブジェクト指向特有の効果が現れるか否かである．たとえば，発生する仕様変更に対して，「旧パラダイムに対して対応工数が半減し，かつ，ソフト品質が3倍上がった」という報告が，継続的に聞こえるようであれば，パラダイムシフトは成功したといえる．

別の視点で見ると，昨今のプロダクトはプロダクトラインによる徹底した再利用を行い，見かけ上の開発生産性を向上する傾向が強いので，「パラダイムシフト」後のプロダクトはまるで新パラダイムでうまくいっているかのように装うことができる．つまり上位の経営層から見ると，まるで「オブジェクト指向パラダイムに順調にシフトし，かつ，回収のフェーズになった」ように語ることができるわけである．

さて，本当にオブジェクト指向パラダイムにパラダイムシフトができているのだろうか？

これを知るためには，パラダイムシフト時からのメトリックス分析が必要で，技術的に投資の回収にどれだけ貢献できているかを目標とともに可視化しておくべきである．このメトリックスの測定を怠ると，パラダイムシフト推進者でさえ，努力と苦労をしたわりには，何をやってきたのかわからなくなってしまう危険がある．

とくに，プロダクトラインによる技術の使いまわしとオブジェクト指向による再利用の促進は，それぞれがうまくいき，相乗効果により大規模な開発工数の改善がのぞめるが，オブジェクト指向による効果を割り出し，とくにソフト開発技術による生産性の改善がどの程度できているのかは把握しづらい．

これを正しく測定するには，たとえば，仕様変更対応時の作業内容や作業工数がどのように変化しているか，IE（Industrial Engineering）的な見地から考察したり，変更内容の絶対値が旧パラダイムに対してどのように変化したかを具体的に数値化する必要があり，元来多忙なソフト開発現場にとっては，骨の折れる作業になることは間違いない（**図5**）．

6 投資効果確認のための測定

再利用に関しては，パラダイムシフトした後の最初のプロダクトと再利用のサイクルに入った最初のプロダクトの変更量を管理する必要がある．さらに，同様にN回目のプロダクトと$N+1$回目のプロダクトの比較を続け，要件の変更量に対する作業量

〔図5〕本当にパラダイムシフトができているのか？

〈即効性がある場合〉
目先の作業の効率は9倍！
投資の効果を即座に確認できたが……
同じ生産性を出すには，次の開発でも同じ投資が必要
→パラダイムシフトできていない！

〈即効性がない場合〉
パラダイムシフトすれば，目先の生産性は変わらないが，次の開発で，ノウハウが生きる．また，科学的アプローチが従来よりも少ない投資で効果的に回収する術が開発できる
→パラダイムシフトできている！

〔図6〕オブジェクト指向の効果を確認するためのメトリックス

メトリックスには，開発中に進捗を管理するためのメトリックスと，開発終了時に改善効果を測るためのものがある．それぞれの測定のしやすさや，それが示すことを研究し，測定メニューに入れること

開発中に進捗を管理する	改善効果を測る
要求数，ユースケース数， 外部アクタ数， クラス数，操作数， 属性数，状態数， 遷移数，イベント数 シナリオ数， 例外シナリオ数	集合体クラス数， シングルトンクラス数， 継承された属性数， 継承された操作数， 多重度の関連数， 汎化，特化構造数， クラス結合度， 継承木の深さ

が従来型のアプローチに対して大幅に削減できていれば回収のシナリオが成立する．

注意しなければならないのは，開発プロダクトの合計を従来型と比較することによる整員数の減少などといった大雑把な観点では，現実にかかった工数の比較はできるものの，投資効果の確認はできないということである．つまり，初期投資の効果が「次」や「次の次」のプロダクトに効いてくるわけで，回収という観点からは，長期的な見方をしなければ現実の損益分岐点が見えないということである．

投資回収を考えたときは，たとえば開発工数のみで工数の比較をするという意志がまっとうされれば，正しい評価ができると考えられる．ただし，先に述べたように，プロダクトラインによる共通化とオブジェクト指向の効果は相乗効果として現れるので，この二つを確実に切り分けて考察する必要があることに注意してほしい．

結局，どんぶり勘定では技術の貢献を測り知ることはできないということではないだろうか？　旧パラダイムとの比較によるオブジェクト指向の効果確認方法については，さらに議論を重ねる必要があると認識してほしい(**図6**)．

7 説得材料

組み込みソフト開発の成否は，その業種のビジネスの成否に直結するといっても過言ではない．昨今の組み込みソフトの欠陥による製品の回収劇や企業イメージのダウンによる損失を考えてもこのことは明らかである．このような社会環境の中で，組み込みソフト開発の責任者は，投資回収という視点にQCDS (Quarity，Cost，Delivery，Service)を照らし合わせて，中長期に渡りビジネスを成功に導くための施策を提案していかなければならない．

ここでは，組み込みソフト開発のリーダーが，パラダイムシフトにかかる投資を上級管理職から受けるために必要なバックデータについて考察する．

7.1　パラダイムシフト時の説得材料

初期開発時の投資分は，最初の開発では回収できない．これは，従来の開発手法で成立していた部分に，中長期的な視点で回収率改善のための投資を行おうとしているのだから当然のことだが，中長期的にオブジェクト指向による開発を行い，成果物の再利用を進める中で，初期投資以降の全投資を回収できるポイント(回収計画・損益分岐点)をパラダイムシフトの提案段階から明らかにしておく必要がある．

開発環境の変更による必要投資，教育費に関しては分けて考えるが，パラダイムシフトしなかった場合の損失に対して投資効果があることを明確に示せるよう，パラダイムシフトによる効果を数値化しておく必要がある．この数値はできれば金額に換算しておくことが望ましい．

具体的には，オブジェクト指向パラダイムにシフトすることで得られるすべての効果をリストアップし，項目ごとにどのような効果が期待できるのか，実現したときの具体的な改善値を表明することで，投資の判断を仰ぐことになる(**図7**)．

7.2　再利用時の説得材料

再利用に関しては，プロダクトラインなどによる旧パラダイムでも実現可能なソフトの流用による改善効果と，オブジェクト指向技術による再利用(継承やカプセル化技術など)開発の効率の改善効果を分離しにくい．いわゆる「しゃぶり尽くす」といった開発の考え方，たとえば，マイナーチェンジを繰り返して新機能を商品化するといった思想によるベースラインの共通化(いわゆる流用)と，オブジェクト指向パラダイムによる開発効

〔図7〕オブジェクト指向の効果予測

問題・課題	ソリューション	予測効果	
開発機能の複雑化	モデル化（可視化）	ソフト品質＝従来の2倍	ソフト品質＝従来の3倍
開発機能量の増大	再利用と差分開発の支援	開発生産性＝従来の2倍	開発生産性＝従来の2.5倍
仕様変更量の増大	仕様記述の支援	仕様変更対応期間＝従来の70％	仕様変更対応期間＝従来の60％
開発納期の短縮	再利用と差分開発の支援	フル機能実装まで＝従来の80％	フル機能実装まで＝従来の70％
		Product-1	Product-2

・問題・課題は，できるだけ具体的に示し，ソリューションとの対応付けの論理を固める
・予測効果は従来型の開発に対する効果を具体的に数値で示す
・中長期的に予測効果を算出する

〔図8〕旧パラダイムとの比較

比較項目	算出方法
開発効率	＝開発ステップ数÷かかった工数(人月)＝人月あたりの開発ステップ数
品質	＝発生した欠陥数÷開発ステップ数＝欠陥密度
開発投資	＝かかった投資÷開発ステップ数＝１ステップがいくらでできているか
仕様変更対応効率	＝すべての仕様変更にかかった工数÷仕様変更件数＝仕様変更1件にかかった工数

・同じ指標で算出する
・比較結果を効果としてとりあげる
・旧パラダイムでの開発はターミネートされている可能性があるので，旧パラダイムでの改善トレンドを明らかにしていく

率化の度合い（すなわち改善効果）は分けてとらえる必要がある．このうち，オブジェクト指向による改善効果を精緻に表現するためには，従来型開発を実施した場合の開発効率とオブジェクト指向による開発効率を徹底的に比較しなければならない．徹底した比較をするためには，開発中のあらゆるイベントや作業を追跡し，新旧両方のパラダイムに当てはめて議論する必要がある．

ここで，予測した旧パラダイムで実施した場合の数値を大きく見せればオブジェクト指向パラダイムによる改善効果が大きく見えることになるので，あらかじめ大きめに見積もって評価してしまいたいところだが，技術者の誇りをもって見積もりすることを心がけたい．そうしないと数年後に自分たちの首が絞まることになる．

具体的には，開発生産性の改善率（たとえば，年率15％の生産性改善のトレンド）を加味し，人的要因については，職人的人材が継続的に作業することを前提にして見積もることが望ましい．現実の効果算出の結果，オブジェクト指向パラダイムでの効果が認められないことも大いに考えられる．確認した結果，効果が認められないのであれば，そのドメインの開発にはオブジェクト指向パラダイムを適用する必要はないということである（**図8**）．

オブジェクト指向で再利用の効果を議論する場合，具体的には旧パラダイムの再利用の定義とオブジェクト指向での継承による差分開発の違いやカプセル化による効果の測定ができなければならない．

また，開発プロセスの効果をどう計るか，モデリングによる開発者間や要求元とのコミュニケーションが，昔に比べてどのように改善されたかを測定できなければ，オブジェクト指向やUMLの効果を厳密に議論することはできない．

再利用時にどれだけの効果があったのかという議論をするうえで，旧パラダイムで“再利用や流用”という場合とオブジェクト指向での“差分開発”の違いを明確にすることに加え，副次的な効果をもたらす部分についての考察と効果の測定を工夫する必要がある．

ここで難しいのは，まず，旧パラダイムでの“再利用や流用”による開発効率の改善効果と，オブジェクト指向での“差分開発”による開発効率の改善効果が分離できないことである．たとえばカプセル化の効果を考える場合には，明らかに，旧パラダイムでの流用・再利用に対して納期と品質の面で貢献しているはずだが，数値化できていないのが現状である．

次に，ユースケースのように自然言語で分析し，要求元とのコミュニケーションが改善されている部分で実際にどれくらいの改善効果が得られているか，開発者同士が開発中に仕様や設計方法についての確認や議論をする際に，UMLを利用することで旧パラダイムに比べてどれくらい議論の効率が上がっているかということを数値化する必要もあるということである．つまり，非技術的なデータ要素についても分析する必要があることを認識すべきである．

たとえば，検討工数という比較的あいまいなデータを大雑把に比較することはできても，厳密に旧来の設計資料によるコミュニケーションに対して，UMLのモデルによるコミュニケーションが何時間何分の検討時間を改善しているかということを実際にデータで把握することはとても難しい．

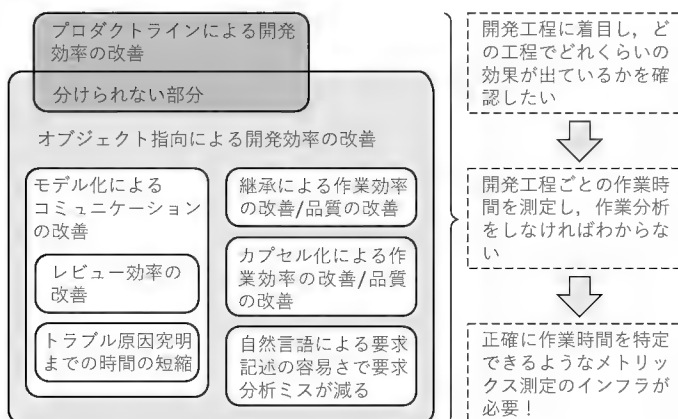
このような工数データなどのメトリックスを採取するためのインフラを構築し，ソフト開発にかかわる作業工数の測定ができなければ，新パラダイムの本当の改善効果は把握できないのである．ソフト開発方法の改善効果をはかるには，IE（Industrial Engineering）的測定をする環境が必要で，開発作業時間を分析するためには，開発者がその行動を簡単に記録し，分析できるようなしくみが必要である．

現在のインフラでも，投資に対する回収の効果を総合的に予測して，計画を策定し，実施してその実績をフィードバックして定式化することは可能だが，それは，あくまでもその開発チームで，その時期（時代）に行ったことにすぎず，変化し続ける組み込みシステムの開発の環境や採用技術の多様性を考えると，実践的な定式化を行うことは困難に近いと思われる．この点に関

しては，学術的な議論や論理立てが不足している部分といえる．

近々の問題としては，とくにオブジェクト指向による投資の回収とプロダクトライン（いわゆる“骨までしゃぶる”ことによる開発の効率化）による投資の回収についてそれぞれの効果を分離して表現することが必要であり，それぞれについて効果を数値化できなければ，胸を張ってオブジェクト指向の効果を語ることはできない（**図9**）．

7.3 投資回収時の説得材料

いままで示してきたとおり，パラダイムシフトを行っている製品には投資効果は現れない．パラダイムシフト後の製品開発において，初めて従来のパラダイムに対する改善効果が現れることを認知しなければならない．また，現実には再利用の最初の製品でも「再利用」技術を利用するのが初めてということから回収のフェーズに移れないという状況がほとんどではないだろうか？

〔図9〕数値化できていない部分

その場合には，パラダイムシフトしている製品で複数のスパイラル型プロセスを実施し，再利用のコツをおさえて，次の製品から高度な再利用を実践することも可能である．ただし，ノウハウを蓄積するためのしくみを運用しながら開発を進めるので，初期投資（パラダイムシフト時の投資）はかさむことになり，全体として効率的に開発できているか否かについて，反省をする必要があるだろう．

また，商品開発を進めながらパラダイムシフトを行う場合には，かぎられた工数でオブジェクト指向による開発に必要な，多様な中間成果物を作成する必要が生じる．そのため，1回の製品開発ではすべての中間成果物を完成させることができない場合もある．この場合は，複数のプロダクトをこなしながら，中間成果物を少しずつ追加し，完全なものに近づけていく方法が現実的である．開発者は自分たちの開発技術そのものに磨きをかけていく努力を忘れてはならない．この努力がパラダイムシフト後のどれだけの製品に必要になるかによっても投資回収のシナリオに変化が起こるわけである．

1世代目にパラダイムシフト，2世代目に再利用技術の習得ができたとすると，現実には同一ドメインの製品ファミリで3世代目の段階で初めて投資回収について実績データを元にした具

〔図10〕投資回収のストーリー

プロダクト1　プロダクト2　プロダクト3
初期投資フェーズ　投資回収フェーズ
＋　－　開発効率　T
オブジェクト指向開発の予測
例.） 開発効率＝開発投資÷ファンクションポイント
A　B　C　E
従来型の開発効率
従来型開発を基準とする
1プロダクト目 パラダイムシフト
2プロダクト目 すでに従来の開発より開発効率は良いが，パラダイムシフト時の不足分の作業が残る
3プロダクト目 モデルが完成し，00のトレンドになる．従来型の約2倍の開発効率！
4プロダクト目 従来型の約2倍の開発効率を維持する

〈シナリオ〉
A＝Bであり，Cからがオブジェクト指向の投資を回収する部分．3プロダクト目からは投資が回収できる
Eが従来型に対するオブジェクト指向型開発の有利分で，同様の組み込みシステムを開発すれば開発しただけ従来型開発に対して有利に開発を進められる

体的な議論ができることになる．その段階では，従来のパラダイムの指標と，パラダイムシフトの最初から3番目の製品での開発生産性および必要なメトリックスの各項目について比較し，投資効果の評価を行うことになる．

1～3世代の比較を行う場合には，担当者の流動による人的側面がとらえにくく，新パラダイムの効果の評価に影響しかねない部分がある．また，メトリックスの検討が十分にできないと，定性的な項目が多々生まれ，かつ，曖昧な定量化で評価指標にされることがある．定性的な評価項目を従来パラダイムとの差異が小さかった場合の追加判断材料にする傾向があるが，可能なかぎり客観的に数値化し，投資回収の評価項目に加えるべきである(**図10**)．

8 流用(プロダクトライン)と再利用(オブジェクト指向)

何度か述べてきているように，投資回収にあたって議論になるのは，流用と再利用の効果に対する評価である．製品開発を実際に行う場合は，大きく流用するほうが開発の効率化にとって有利であり，オブジェクト指向パラダイムの開発効率化に対する効果は，流用のそれに比べると，小さく，見えにくい．

たとえば，ソフトの変更を行うときに，ホットスポットの特定に合致し，フック部分の変更で対処できたのであれば，オブジェクト指向パラダイムは十分に開発対象の制御の特徴にかなった開発効率化に貢献しているといえるだろう．

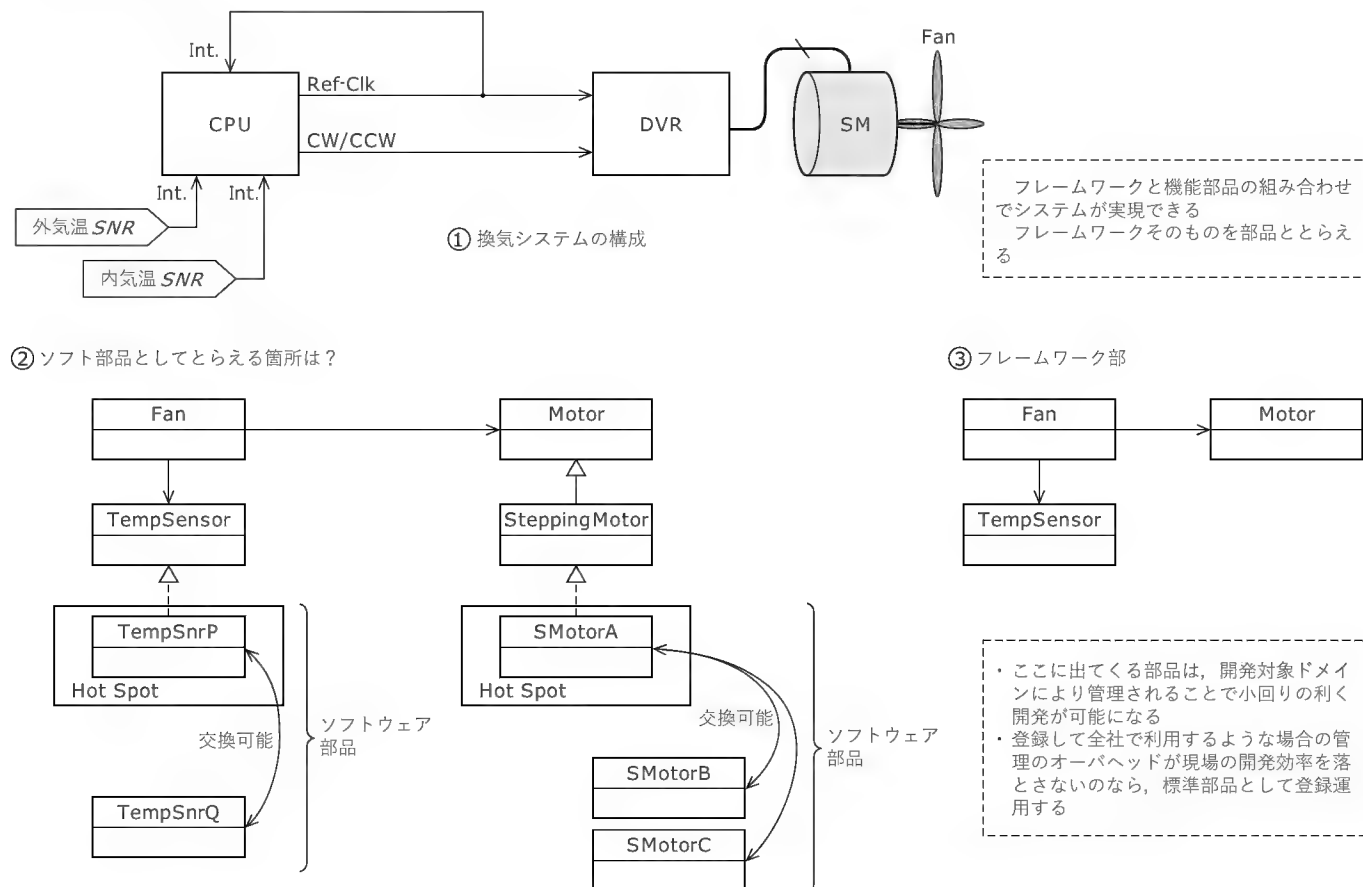
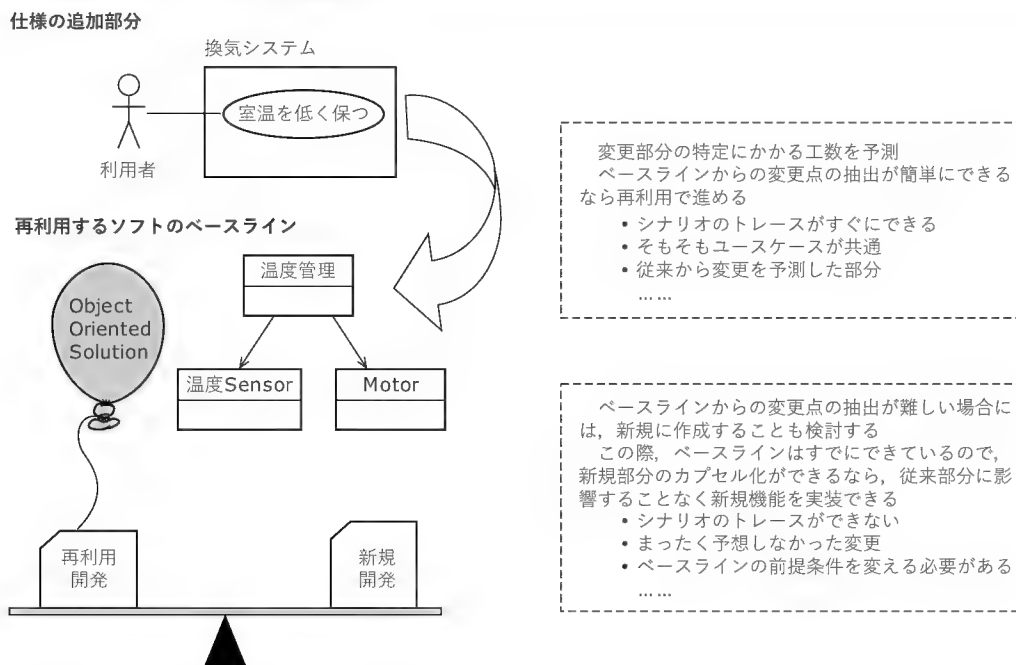
また，ソフトの変更を行うときに，変更点をユースケース図から特定し，ユースケースシナリオの変更から，変更が必要なクラスを特定し，変更部分の影響をクラス図の関連から特定し，変更作業から検査までの各作業を最小の作業量におさえて高品質のソフトを提供できるのなら，オブジェクト指向パラダイムは十分にその制御対象の特徴に合致した開発の効率化に貢献しているといえる．

変更に対する作業の方法が，インクリメンタルに行われなかったり，せっかくオブジェクト指向パラダイムに沿ったモデルがあるのに従来のパラダイムと同様にソースコードのみで変更に対応していたのでは，オブジェクト指向パラダイムによるソフト開発効率の改善効果は最大化できていないということができる．

9 部品化による開発効率の限界

オブジェクト指向によるソフト開発効率の改善というと，短絡的にソフトの部品化による再利用を想像する傾向があるが，本来ハードとは違った属性のものにハードの管理を適用するに

〔図11〕部品化の考え方

・ここに出てくる部品は，開発対象ドメインにより管理されることで小回りの利く開発が可能になる
・登録して全社で利用するような場合の管理のオーバヘッドが現場の開発効率を落とさないのなら，標準部品として登録運用する

〔図12〕再利用するか新規開発するか，どちらが効率的か

は無理がある．

部品化による開発効率の改善には限界（頭打ち）があり，具体的には部品の登録，再利用のための管理などのメンテナンスがオーバヘッドになり，メンテナンスを最低限の作業にとどめることができたとしても，部品となるソフトウェアは技術の進歩や変化とともに増えるので，ソフトの部品化による効率改善に限界があるのは明白である．

また，共通に利用できるようにするということは，部品自体の汎用性を高めることになる．このため，本来高い付加価値をカスタマイズすることにより，提供する機能の変化に対する柔軟性を期待されているソフトに，ハードで行う部品化の発想を適用することになる．これでは，ハード的発想の責務範囲でソフトを作るようになってしまい，システムに最適なアーキテクチャを提供できなくなる．

結果として，再利用できない部分は払拭できず，新規開発を行うたびに手書き部分に工数が取られ，手書き部分を新規に作成することで，そこに関する品質レベルは従来と変わらず，品質の作り込みから検査工程までの開発効率を改善できないという状況が起こるだろう．

つまり，部品化を進めるということは，本来異なる問題領域ごとの対処の特化・最適化に対して柔軟に対応する役割をもった「ソフトによる実装の優位性」を阻害してしまう．すべてのサブモジュールに共通な標準部品としてソフト部品を管理することは，組み込みシステム開発の効率化に貢献するとは考えにくい．

汎用的なソフトの部品化よりも，組み込みシステムごとの問題領域に特化したフレームワーク（ホットスポット・フローズンスポットの抽出による対象ドメインのモデル化と変更部分のみを部品化する）による開発を推奨する．

細分化されたCPUごとの各サブモジュールはフレームワークをもち，そのフレームワークに対する部品を利用することで，制御対象となるハードウェアごとにソフトの部品化が起こる．それぞれのシステムに対応した再利用が柔軟に実装できることで，QCDS達成に貢献することになる．

このように，サブモジュール自体が一つの大きな部品となり，サブシステムごと部品として登録されるのが自然ではないだろうか（**図11**，前頁）．

部品化，共通化をしたというと，唯一無二の最適解を得たかのように考えがちだが，現実には部品化することによる開発効率のダウン，共通化することによる柔軟性，仕様変更対応にあたっての対応作業効率のダウンが起こっている．つまり，基本的に再利用技術は難しく，開発効率を考えると新規に開発を行ったほうが良い場合もあるので，投資回収の観点からしっかり議論したうえで，新規に作成するのか，再利用をするかを決めることが望ましい．

開発者や開発チームは十分な議論をせずに流用率・再利用率を管理パラメータにしがちだが，よく議論し，確立した見積もり技術を駆使し，再利用した場合と新規に開発した場合とをそれぞれ比較し，よりロスの少ないほうを選択するのが賢明といえるだろう（**図12**）．

すぎうら・ひでき　富士ゼロックス（株）

第5章

現実的な開発効率向上を考える

どうして組み込み分野でオブジェクト指向が広まらないのか

杉浦英樹

開発環境の整備が進み，実際の商品開発に適応した例が少しずつ出はじめているものの，組み込み分野の製品開発がオブジェクト指向を中心に進められている状況とはいいがたい．本章では，なぜオブジェクト指向による開発が，組み込み製品開発の中心にならないのか，その点について考察する．

（筆者）

1 開発環境がないので新規投資が必要

オブジェクト指向パラダイムに必要な組み込み型ソフト開発のための開発環境は整備されつつある．現実に，組み込みソフト開発における技術的な障壁は減っており，5年前の開発環境に比べるとオブジェクト指向の導入のしやすさは雲泥の差である．

ところが，新規に投資を提案する開発現場では，パラダイムシフト時に発生する開発環境や教育に対する投資，一時的な開発効率の悪化といった懸念から，投資価値があるかないかの判断に苦しみ，いざ提案というところでも二の足を踏む場面が容易に想像できる．

パラダイムシフト前の開発現場で，「いままでのパラダイムである程度の品質のソフト（ファームウェア）が提供できており，開発プロセスや組織が今のパラダイムに合致する形で維持できている」，かつ，「多少の残業はあっても，現状を維持することで中長期的に見て安定した品質の組み込み製品を継続的に市場に投入できる見込みがある」ので，「あえて新たに投資をして開発効率を改善しなくても事足りる」という考えをもつことは，とても自然で現実的といえる．

そのような会社環境に置かれ，開発している現場技術者の方々に対して，あえてオブジェクト指向パラダイムを推奨するには，従来のパラダイムとは異なる部分，つまりオブジェクト指向開発を使うことの「旨み」に着目した，はっきりした方向性を提示する必要がある．

たとえば，リアルタイム性を悪化させるようなリスクをおかしてまでオブジェクト指向パラダイムにシフトする理由は何だろうか？　現状の開発結果としてのソフトにとくに問題がないのなら，将来に起こることが予測できる課題を整理することで，その課題に対処するためのパラダイムの一つとしてオブジェクト指向をあげるのは一つの手である．

しかし，その際に注意しなければならないのは，課題が具体的になっており，オブジェクト指向の提供するソリューションが明らかにその課題に対する解決策としてあてはまることが検証できているということの確認を怠らないことである．当然だが，オブジェクト指向パラダイムにシフトするための投資が従来のパラダイムで対策するよりも効率的にまたは普遍的にその課題を解決するということにつながるなら，オブジェクト指向を選択する際の説得材料として十分ということができるのではないだろうか（**図1**）．

開発環境や教育の投資についても，課題に対する対応としての投資回収のシナリオ作成に取り込み，「長期的に見て，今投資をすることが将来のリスクを回避する」ということを明確に示唆する必要がある．

つまり，開発現場の担当者からボトムアップで，「従来のパラダイムにありがちな場当たり的開発からオブジェクト指向などの工学的開発手法にパラダイムをシフトするための提案をする」ということは，開発対象の問題領域に対して中期・長期的な改善のビジョンをもち，具体的にパラダイムシフトした場合としない場合の状況を予測し，投資に対する採算性を加味した計画を作れなければいけないということになる．

その意味では，提案者にはそのドメイン（問題領域）の専門家（プロフェッショナルリーダー）としての知見が必要で，そのプロフェッショナルリーダーが先にあげたような課題をドメインの

〔図1〕オブジェクト指向化にともなう投資を引き出すための説得

問題 課題	従来の対策	オブジェクト指向の対策	評価
A	解決できない	解決の見込みあり	一本！
B	解決できるが，とてもたいへん	従来よりも簡単に解決できそう	有効！
C	解決できるが再発防止まではできそうもない	再発防止を含む抜本的な対策ができそう	効果！

課題として認識できなければ，業務改善としてのパラダイムシフトは始まらない．

もう一つの見方として，周囲の技術をオブジェクト指向で固めていく方法（たとえば，接続するクライアントがオブジェクト指向を利用しており，その思想を効率的にサービスに結びつけるためにはオブジェクト指向による開発がいちばん効率的であるという論理）もあるが，対象ドメインのプロフェッショナルリーダーを説得できなければ，一蹴されることになるだろう．

また，開発対象のメンテナンスサイクルの負荷が大きく，開発現場の技術に対するモラルが低い場合には，現場技術者（開発担当者）のスキルアップやモラルアップをねらってオブジェクト指向技術を採用することもパラダイムシフトの理由として認知できるはずである．

いずれにしても，オブジェクト指向技術を正しく理解した先導者が担当者に対して指導をしながら推進しないと，開発現場の中途半端な技術解釈で作業が進み，結果としてパラダイムシフトが完成しない．それどころか，へたをするとパラダイムシフト終了後に確認した生産性の指標から，「パラダイムシフト前のほうが，開発効率が高い」という測定結果が出てしまい，オブジェクト指向技術が誤解されたまま，「効果なし」というレッテルを貼られてしまうことにもなりかねない．

このことは，技術として間違っていないにもかかわらず認められないという状況を作り出し，結果として効果を出している開発チームにも悪影響を与えることになってしまう．ソフト開発全体としてこのような誤解を起こさないために，開発チームには技術の正しい認識と解釈，運用が求められる（**図2**）．

2 リプロダクションが多く，過去の資産を流用するケースが多い

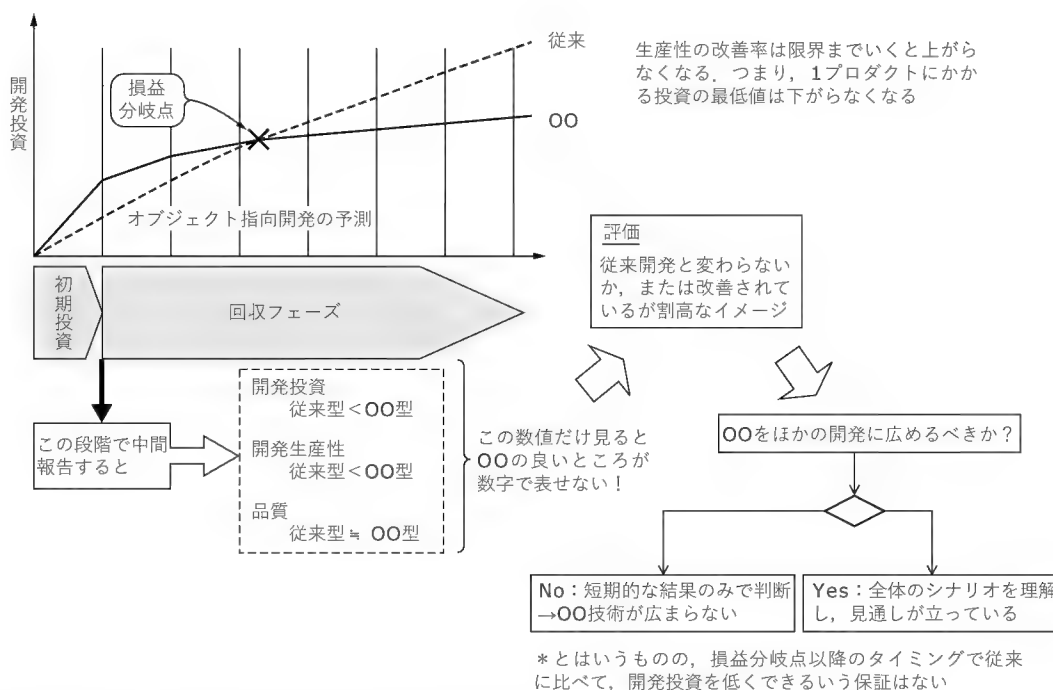
組み込み型システムは，型を起こした後は，なるべく型変更しないで，ソフトだけを入れ替えることにより，機能の追加や変更をするように考える．一度型を起こした製品が市場に投入されれば，その後は製品版のソフトをベースラインにした変更開発，つまり保守サイクルが始まることになる．

オブジェクト指向にパラダイムシフトするなら，前述の現パラダイムでの課題の明確化とともに，新規開発（型を起こすような開発）に合わせてパラダイムを変えられるように準備する必要がある．

オブジェクト化する際には，最初からすべてのソフトを対象とせず，部分的に適用するのも手である．が，従来のパラダイムとオブジェクト指向パラダイムを同時に開発管理することは，パラダイムシフトにとっては足かせになるので注意が必要である．

たとえば，オブジェクト指向開発プロセスを定義して，それにしたがって開発を進めるチームと，従来型の開発プロセスにしたがって開発を進めるチームを同時並行して管理することは，両チームを完全に分ければ実現できそうである．しかし現実には，担当者は担当者同士の情報網をもっており，その情報網で情報交換しながら商品開発を進めているので，両方のプロセスによる品質確立までの方法を十分に理解できていないかぎりは担当者レベルでの混乱が生じることになる．

また，小規模な開発では，同じ担当者が複数のパラダイムを利用して同時に開発を進める可能性もあるので，開発の管理が複雑化する．現場のフロントラインマネージャは複数のパラダ

〔図2〕パラダイムシフト時の誤解によるオブジェクト指向技術のダメージ

イムによる開発を意識して，ソフトの開発計画を十分に検討する必要がある(**図3**).

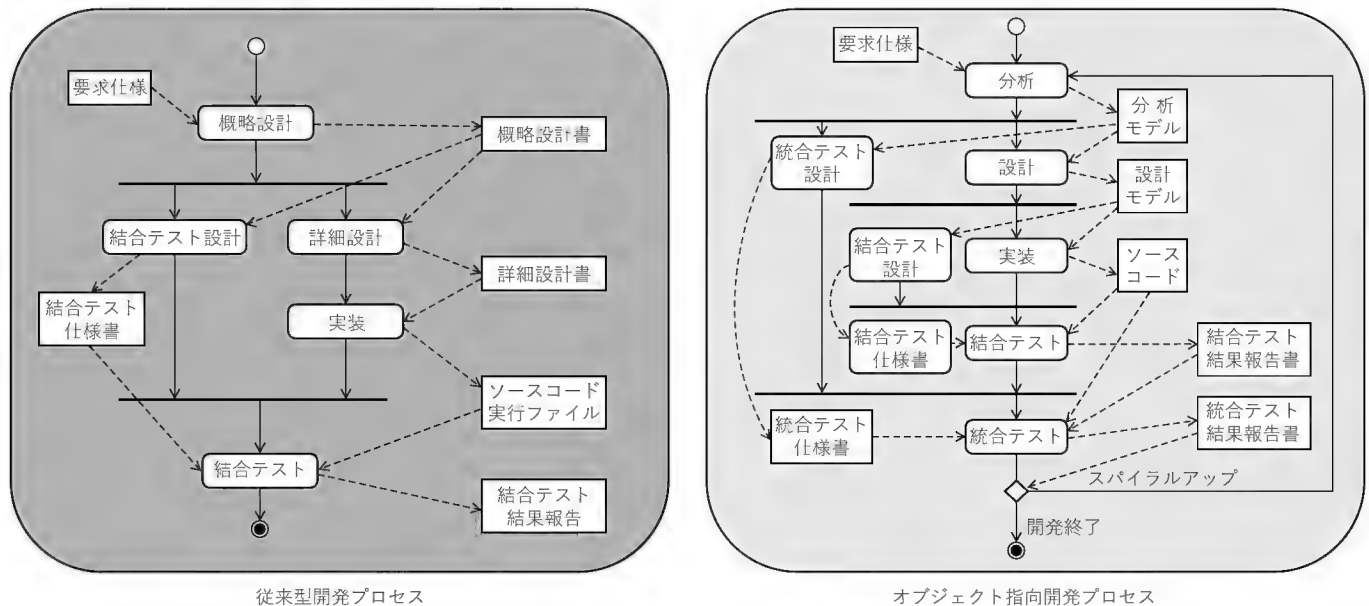
3 「Virtual」が難しい

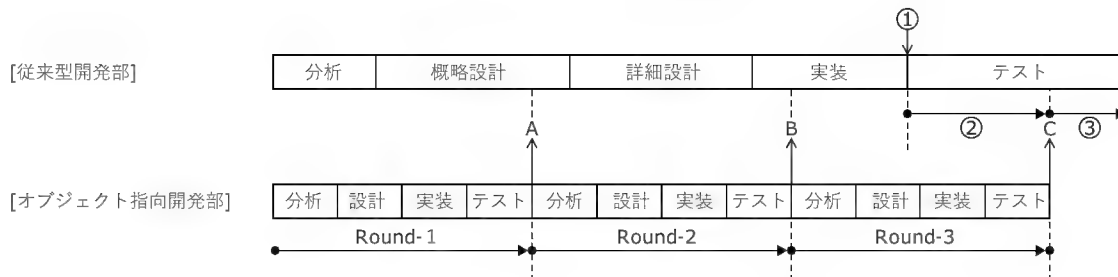
オブジェクト指向が再利用を効果的に行うために支援している技術の一つが継承機構であるということができる．ところが，組み込みソフト開発現場では，この継承機構がなかなか有効に利用できていないのが現実である．とくに組み込み製品では，限られたメモリ容量と限られたCPUのパフォーマンスでコンパクトに制御を行う必要があるため，“New”や“Delete”といった動的なメモリの生成や削除をしないことが基本になる．コンストラクタはパワーオン時に集中させることで，電源が入った後のリアルタイム性に問題が発生することを抑制する．

ただし，昨今の組み込みシステムへの要求では，省エネルギの観点からパワーオン時の初期化時間への制約が厳しくなってきており，今後は初期化処理を高速で行うためにどうするかということが課題になるだろう．このように，ユーザーからの要求と実装技術の不整合が起こる場面も多いのである．

組み込み型のソフト開発では，開発したソフトをROM化する際には，動的なプログラムコードの生成はRAM上で行うことはしないで，ROM上に静的にプログラムを配置する．仮想クラスや仮想関数を実装する場合には，設計の段階から実装方法を考慮し，実装ファイルを作成するまでにROM，RAMでのプ

〔図3〕従来のプロダクトラインとオブジェクト指向

従来型開発プロセス

オブジェクト指向開発プロセス

開発管理者は[従来型開発プロセス]，[オブジェクト指向開発プロセス]という両方のプロセスを管理しなければならない！

オブジェクト指向による開発部分については，短いサイクルで分析からテストまでを完了できるが，単体テスト後にシステム結合テストを行おうとしても，従来型の工程では実装に至っていないのでテストできない．

上記のようにAのリリースはせずに，Bでまとめてリリースし，①のテストはBのリリースで開始する．その後のCのリリースは，①のテストにカットインするように計画して対処する．

よって，②のテスト期間中はA，Bでリリースした機能に限定したテスト仕様でテストする．Cでリリースした機能は③でテストする

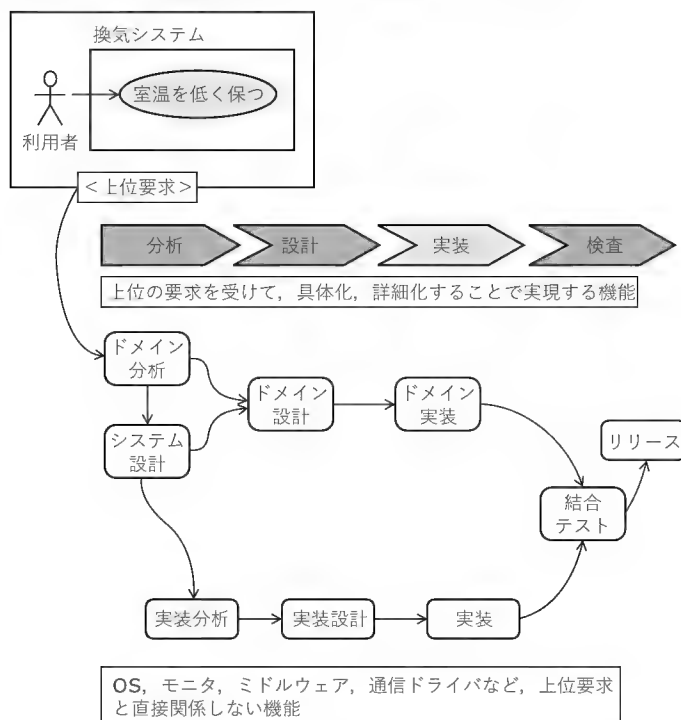
ログラムの動作をシミュレーションするなどの工夫をして，モデル要素とプログラムの関係を明確にする．これはプログラムの品質を開発の初期から意識し，実装のためにはどのような考え方やソフトの構造が必要になるかを考察するためには有効な方法である．

組み込みソフトの実装に関する技術は，分析から設計までの技術とは異なり，開発対象ドメインとは切り離して考えることができる．したがって，実装時の制約に関しては機能要件から始まる分析/設計とは切り離して考えられるように，開発プロセスを定義するべきである(**図4**)．

パラダイムシフトできているのかいないのかに着目して現実の開発をみてみると，仕様変更が起こった際の開発担当者の作業内容でパラダイムシフトできているか否かの判断が可能である．

たとえば，あるクラスの操作に変更を加える必要がある場合，そのクラスやメソッドを仮想関数にしてから，子クラスで再定義することにより実装できているだろうか？　パラダイムシフトができていない担当者は，間違いなく仮想関数による対応作業は行わず，直接，すでにあるクラスの該当する操作を変更する．

つまり，仕様の変更が発生した場合に，元のソフトに対しての変更部分を再定義部分として独立して扱う(差分開発)ことによる利点が理解できていないことが大きな理由といえる．それに加え仮想関数をよく知らない，または，よく知っていると勘違いしていることにより，差分開発終了後の洗練までの作業を一気に進めたような気になる．これが間違いになる．そのような担当者に限って，なぜそのように仕様が変更されたのかといった技術背景，どうしてそのように変更したのかといった改訂

〔図4〕ドメイン領域と組み込みソフト実装に関する技術の違い

履歴をとるような几帳面さを持ち合わせていないので，開発の後半や開発終了後に作業した内容についての反省や今後の開発へのフィードバックができなくなってしまうのである．

テストまでの工程を含めたソフト品質のパフォーマンスを考慮した場合，オブジェクト指向パラダイムの提供するソリューションがもっとも効果を発揮するということを理解している担当者はどの程度いるのだろうか？　また，理解できていたとしても，実践できている者はどれくらいいるだろう？　開発の現場で組み込み型ソフト開発を推進しているリーダーは，率先してオブジェクト指向パラダイムの有利な点や不利な部分を勉強し，開発現場の開発効率の改善とともに，現場開発者のスキルアップを図るべきだろう．

たとえば，仮想関数については，ソフト品質の観点と検査までのすべての作業工数の観点から使用方法を研究し，開発対象の制約を含めて運用ルールを規定すべきである(**図5**)．

4 従来のパラダイムで十分に品質や生産性が取れている

従来のパラダイムで満足している場合には，本当にオブジェクト指向にパラダイムシフトする必要があるのかどうかを，考え直す必要がある．検討する際には，まず現パラダイムによる開発の問題と将来起こるであろう課題を整理して，解決策とオブジェクト指向パラダイムが提供するソリューションを比較してみればよい．問題や課題のすべてがオブジェクト指向のソリューションで解決できることはないだろう．それでも，従来のパラダイムに比べて開発効率が改善される見込みがあるなら，挑戦してみるのがポジティブな技術者ではないだろうか？

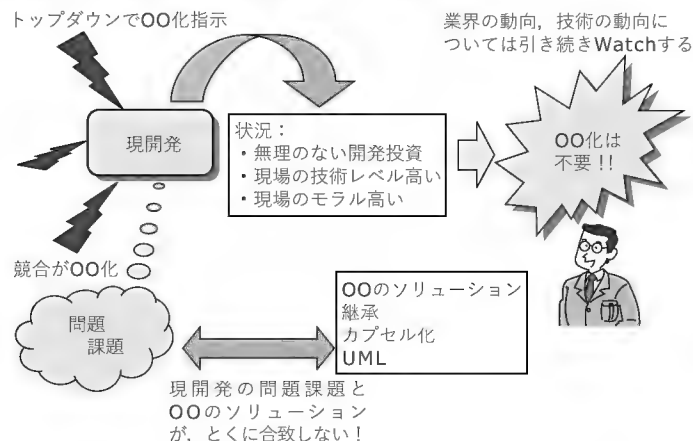
重要なのは，従来の開発でうまくいっているところにあえてオブジェクト指向のパラダイムを適応する必要はないと英断する気構えであり，時流に流されない頑固さも必要かもしれない．

たとえば，そこにハサミがあるからといって，いままでナイフで切っていたものがすべて切れるとはかぎらない．ナイフ(従来のパラダイム)を利用したほうがハサミ(オブジェクト指向パラ

〔図5〕継承の運用ルール

継承運用ルール
ベースラインが特定されたプロダクトについては，以下を厳守すること
① ベースラインの変更点は，既存ユースケースにより特定する
② 特定したユースケースシナリオから，シーケンス図を確認し，さらに，関係するクラスインスタンスを特定する
③ 特定したクラスインスタンスに対して，必要な変更をリストアップする
④ 特定したクラスが仮想クラスでない場合は，仮想クラス宣言する(ホットスポットの登録)
⑤ 特定したクラスに子クラスを関連付け，クラスに対する変更分を子クラスに再定義する
⑥ 親クラスへの変更部分の取り込みは，タイミングスペックが守れない場合とメモリ容量上の問題がある場合を除き禁止する
⑦ ⑥に該当し，洗練を行う場合は，関係者によるレビューを開催し，技術的に問題がないことを確認する……
……

〔図6〕パラダイムシフトしないという「英断」

〔図7〕測定項目を抽出し，分析し，定式化することで，より良いQCDSの獲得をめざすのがエンジニアリング

ダイム)を利用するよりもはるかに効果的な場合もあるということを忘れてはならない(**図6**)．

5 ソフトウェア開発がエンジニアリングになっていない

最近のソフトウェアに関する雑誌や書籍を見ると，ソースコードが少ないばかりか，数値データによる議論がまったくといっていいほど行われていない．先般，あるパネルディスカッションで，受講者が「われわれはエンジニアリングをしたいのだが，このような内容では文学になってしまう」と嘆いておられた．

われわれソフト開発エンジニアは，名前のとおりエンジニアリングを進めるために生を享けたといっても過言ではないだろう．それならば，ソフトウェアの開発をエンジニアリングに近づけるためにはどうしたらよいのだろう？

ソフト開発における数値化が難しいのは，担当者の行動や考え方に依存する部分が大きいためで，開発作業そのものを正しく数値化して分析しなければ開発生産性の改善は測れない．また，最終成果物を測定して評価することは可能だが，そこからはそのソフトの性能や機能に関する評価結果としての数値は得られるものの，できあがるまでのプロセスや，かかった時間についての関連は分析できない．

エンジニアリングを行うためには何を，何のために測定するのかについての検討を十分に行い，測定するメトリックスを抽出する．ここで抽出された数値データを元に，開発するソフトウェアの特徴や特性を把握できる．メトリックスの種類としては，開発期間を通して定期的に測定するものと，開発終了時にまとめて測定するものの2種類があり，前者は開発を管理する指標として，後者は開発したものの技術的評価を行うものととらえる．実際に測定を行って得られるデータの分析を行い，分析結果から可能ならそれぞれのメトリックスの関係や，時系列での変化を数式化し，今後の開発においてその式を利用することで開発生産性や得られるQCDS(Quality，Cost，Delivery，Service)の予測精度を向上したり，開発管理の効率化に結びつけたりすべきである．

何をどう測定し，何に反映すればエンジニアリングと言えるソフト開発ができるのだろうか？　これは，現場開発者および開発推進者が日々の開発作業の中でいつでも考えていかなければならない課題だと思う(**図7**)．

6 メトリックスの具体的な検討

6.1 Q(品質)

たとえば，ソースコード1000行あたりの抽出欠陥数を考えてみる．このデータは，欠陥を抽出したテストのテスト品質に依存する．したがって，どのような品質のテストをどれだけしたかということを考慮しなければならない．つまり，テスト品質の測定ができなければ欠陥密度の計画や実績の確認はできない．また，テストで発見された欠陥は，プロセス上本来発見されるべき工程で発見されたのか否かについて，開発終了後に分析し，次に開発を行う際のプロセス改善や検査内容の改善活動へフィードバックする．

開発手法や採用技術の効果を計るには，どれくらい複雑なシステムをどれくらい簡単に作ったかという指標が必要になる．たとえば，(ファンクションポイント)÷(ソフトウェア複雑度)で示される数値が大きいほうが，シンプルな良い実装をしているということができる．モデルやソフトを単純化することにより，開発工程のすべてに対する負荷が大きく緩和される．また，品質の確立のための活動に対する負荷を減らすことにもなるので，ソフト品質を確保するためにも非常に重要な指標になると考えられる．

継承を利用して差分で開発したか，直接クラスのメンバとしての操作を編集したか，直接クラスのメンバとして新たに操作を追加したか，新たにクラスを追加したか，これらのいずれかの活動にともない，欠陥がどのように発生したかということと

関連付けることで，それぞれの実現手段を見きわめることもできる．直接操作を編集しても，差分で開発しても，得られる品質が同じなら，仮想関数によるメモリの消費(バーチャルテーブルによるROM，RAM容量の消費)が少ない方法であるメンバ操作を直接編集するのが効果的ということもできる．つまり，組み込みソフト開発においては，仮想関数による差分開発を最優先すればよいということではないということができる(**図8**)．

6.2 C(コスト)

ソフト開発におけるコストは使用するCPUやメモリの量産価格で代替されるが，はたしてそれがソフト開発におけるコストといえるのだろうか？ ソフト開発におけるコストは，開発費という観点でとらえるべきではなかろうか？ 企業の開発活動の指標は，いろいろな定義がされているが，結局は投資回収の効率を明確に提示して，これを指標としてとらえるべきである．

たとえば開発環境に対して投資をする場合に，元の取れない(採算性の合わない)設備にいくら投資をしても現実にその設備を利用した開発活動を維持・継続することができなくなってしまったのでは意味がない．売れる製品を開発し，製品を売った利益で開発環境を改善し，改善したことによりさらに効率よく売れる製品を開発するという良いサイクルを意識して作らなければ，開発現場が最大限に機能することはない．

開発費をとらえるときには，直接開発費としての人件費，開発環境費，教育費，管理のための活動にともなう人件費などをすべてもらさず計算しておく必要がある．可能なかぎり開発費は具体的に金額換算し，開発現場としても，どれほどの投資を受けているかを自覚して開発を進めるべきである．

もちろん，CPUの選択やROM，RAM，NVMなどの使用量を管理することで，製品にかかる直接的なコストの管理を行うことを忘れてはいけない(**図9**)．

〔図8〕品質を計るためのメトリックス

項目	算出方法
ソフトウェア品質	欠陥密度
テスト品質	テスト効率＝抽出欠陥数÷テスト項目 または，抽出欠陥数÷テスト時間
方法論の品質	シンプル率＝ファンクションポイント数÷ソフト複雑度 要求数，ユースケース数，外部アクタ数，クラス数，操作数，属性数，状態数，遷移数，イベント数，シナリオ数，例外シナリオ数，集合体クラス数，シングルトンクラス数，継承された属性数，継承された操作数，多重度の関連数，汎化，特化構造数，クラス結合度，継承木の深さなどのメトリックスによる基本測定

〔図9〕開発投資と回収を計るためのメトリックス

開発投資を計るメトリックス	回収を計るメトリックス
教育費　教育工数 　　　　講師費用，セミナー費 直接工数　分析工程 　　　　設計工程 　　　　実装工程 間接工数　検査工程 管理工数　会議打ち合わせ 　　　　CMM-RM 　　　　CMM-SPP 　　　　CMM-SPTO 　　　　CMM-SCM 　　　　CMM-SSM 　　　　CMM-SQA 設備投資　CASEツール 　　　　開発環境費 　　　　　Emulator 　　　　　ICE/Debugger 　　　　　PC/WS	① 製品利益に占めるソフトの割合 ② 販売量 ③ 販売利益 回収は，①×②×③で示される 現実に難しいのは，①の決定 たとえば， 製品＝メカ＋電気＋ソフトとして，それぞれの開発に投入した投資率をそのまま，利益に占める割合として算出する 開発費比：メカ，ソフト，電気 型(ハード)変更がなければソフトがすべてを回収することになる ただし，この場合，従来開発に対してソフト開発効率が上がった分，製品利益に占めるソフトの割合が減ることになるので注意

6.3 D(納期)

納期を制御するのは，要求の的確な分析と，それに対する正確な工数の見積もりである．安定した(担当者の流動の少ない)開発チームで納期管理をするためには，経験則を中心としたPDCAをまわす．前提とした要求機能や要求性能，制約条件などに変化があればPDCAのサイクルにのっとり，そのつど計画を見直す必要がある．また，可能なかぎり工学的手法による見積もりの結果から開発計画を作成し，作成した計画と実際の開発との乖離について，短い周期(通常週に1回程度)で定期的にチェックし，乖離した場合にはその原因を究明するとともに，修正するスケジュールに前提条件の変動要素を反映する．乖離要因の特定やそれに対する対処方法については，実際に起こったことをノウハウとして蓄積し活用する．

具体的に必要なのは，要求の管理情報，その要求に対する作業工数見積もりの前提条件，見積もり結果を前提にした開発計画と実績の追跡グラフである．グラフの縦軸は，旧パラダイムから使用している開発ステップ数を利用してもよいし，それ以外の中間成果物の作成量を管理することで測ることも可能である(**図10**)．

7 電気屋さんもオブジェクト指向を使いましょう

7.1 IPとオブジェクト指向

すでにC言語でのASIC開発の事例が出はじめていることは周知のとおりである．IP自体の整備はCAEにおいて十分に浸透してきており，いわゆる部品化と部品化された部品を使用することによる製品開発は実用の域に達している．

言語による記述，部品化とくればまさにソフトウェアの開発と同様であり，ソフトウェア開発のアプローチが必要になることは間違いない．オブジェクト指向によるASICやFPGAを構成する要素の再利用は，今後数年のうちに実用化できるようになると考えられる．

具体的には，IPをオブジェクトとしてとらえ，関連を定義することで，ソフト/ハードを意識しないで接続することが可能になる．実装するにあたっては，その処理機能をソフト(CPUが処理する形)で実装するか，ハード(専用の回路を設けて処理する

形)で実装するかの切り分けを定義するとともに，ハードで実装する場合には，ソフトとのインターフェースの方法を明らかにする．

ソフトからハードのサービスを利用する場合には直接関連を引くことで，定義された手続きを操作に記述すれば簡単に利用可能になるだろう．ハードがソフトを利用する場合には，割り込みなどの内部イベントを利用するなど，ソフトに通知するための機構が必要になる．つまり，ハードで処理を行う場合にはソフトとのインターフェース機構までを含めて回路を組む必要がある．たとえば，ハードの処理が終了するとあるレジスタの数値が書き換わるように組んであるなら，ソフトはそのレジスタを定期的に監視することでハードの処理の終了を確認することができる．

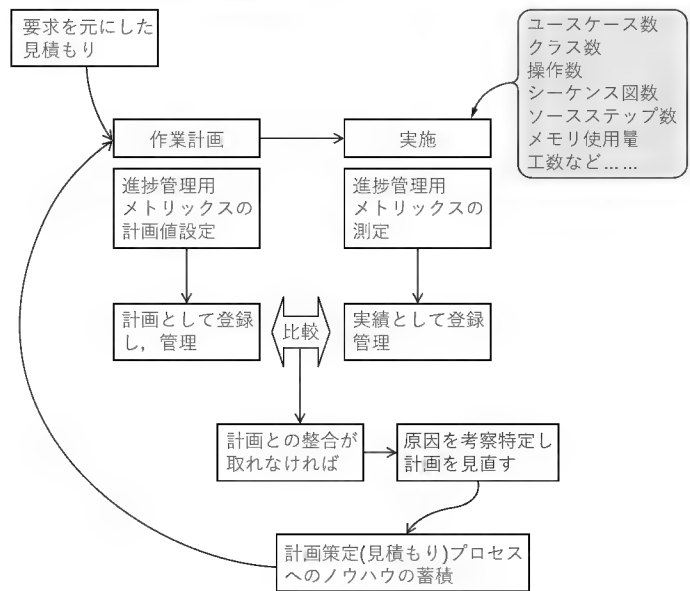
7.2 ハードでもソフトでも実装可能

ハードでもソフトでも実装が可能になることで，何が良くなるのだろうか．まず，組み込み型の製品を考えた場合には，機器の容積を小さくしたり，制御時の信頼性を高めたりするために，専用のASICを開発して実装することが一般に行われている．ASIC自体は毎回作り直すごとに投資がかかるため，ソフトとの連携，すなわち組み込み型の制御アーキテクチャをどのようにするかということが課題としてあげられる部分である．

ASICそのものも製造プロセスや採用技術からチップ面積，つまり実装する回路数に制約がある．ということは，一つの組み込み機器における実装機能数が増大していけば，当然，ASICに入れられなくなる機能が出てくるわけである．そこで，とくにハードで実装しなくてもよい部分，つまりソフトで実装しても事足りる部分を明確にしておけば，ハード実装量の制約から，ソフトで実装する部分が容易に抽出できることになる．

一方，別の視点で考えると，組み込みシステム全体をとらえた場合には，制御が確立して安定している部分(すなわち，オブジェクト指向でいうところのフローズンスポット)やリアルタイム要求の強い部分はハードウェアで実装し，柔軟性や拡張性が求められる部分(すなわち，オブジェクト指向でいうところのホットスポット)はソフトで実装することが望ましいということは一目瞭然である．

もちろん，実装コストの問題は避けて通ることができないので，ソフト開発者も電気回路の実装コストに関する感覚を磨いておく必要がある．つまり，採算性，投資回収を考えて実装技術選択の基準を設ければ，判断に迷って，仕様確定時期を遅らせることがなくなるのではないだろうか？ このようにしてシステム内部のアーキテクチャを構築する場合は，開発対象となるドメイン(問題領域)の組み込みシステムとしてのシステム分析を実施し，フローズンスポットとホットスポットを抽出すれば，ハードウェアの実装部分とソフトウェアの実装部分が簡単に特定でき，最適解としてのシステムアーキテクチャが確立しやすくなる．理想的には，組み込みシステムのハードウェアは変更しない部分，ソフトウェアは変更する部分だけを実装し，協調

〔図10〕デリバリの管理要素

することで組み込みシステム製品として機能する．これが，本来のハードとソフトのあるべき分け方ではないだろうか(**図11**)．

7.3 ソフトでもハードでも分析は一緒

組み込みシステムに求められる機能・要件は，ハードウェアとソフトウェアの責務範囲に分担され，それぞれの責務の中で実現していくことになる．したがって，組み込みシステムに要求される要件の分析は共通であり，これをハードウェア担当者と共有することで従来のパラダイムによく見られたハードウェア担当者とソフトウェア担当者のミスコミュニケーションが改善されるわけである．

ハード担当者とソフト担当者が協力して開発対象の組み込み機器の要件を分析し，ともにハードとソフトの責務範囲を特定し，シミュレーション(検証)し，それぞれの開発作業のインプットにするのが本来あるべき組み込み開発の姿ではなかろうか？

現在のパラダイムでは，こと製品に関する決定権(あるいは主導権)がハード担当者側に偏りすぎる傾向があるので，この偏重を是正する試みとして開発者全員がオブジェクト指向パラダイムによる開発を進めることを提案したい(**図12**)．

8 それでもパラダイムシフトできない現実の開発

オブジェクト指向を組み込みソフト開発現場に導入した後で起こる症状を考察する．ここで取り上げる症状はいずれも，パラダイムシフトでねらったことと違う状況になっているものである．一体何が原因でこのような状況になるのかを推察し，どうすればあるべき姿に近づくことができるかを考えたい．

8.1 症状-1：デバッグ時の行動

各モジュールの結合デバッグ(結合テスト)時の行動が腑に落

〔図11〕ハードでもソフトでも実装できるようにモデルを作成する

ソフトによる実装

Fan

TempSensor
GetTemp()

TempSensor
GetTemp()
ホットスポット

ソフトウェア部品

継承を用いれば，子クラスで再定義されるので，TempSnrPのGetTemp()の操作の処理が行われる

ハードによる実装

Fan

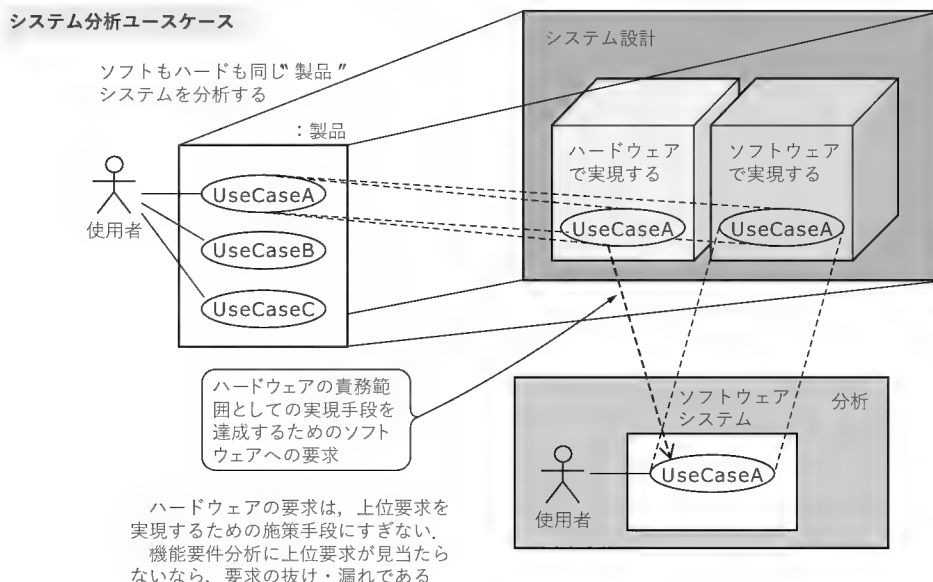
TempSensor
GetTemp()

Hardware
ホットスポット

ハードウェア部品

ハードによる実装では，親クラスの操作がハードとのインターフェース処理を行う

ホットスポット部分の実装をハードでもソフトでも実装できるように工夫する．具体的には部品となるクラスに対するインターフェースを親クラスに定義し，子クラスがソフトなら，再定義したメソッドが処理として機能する．子クラスがハードなら，親クラスに定義したハードとのインターフェースが機能し，ハードが処理を行うことで対処できる

〔図12〕システムを分析するユースケースとソフトとハードの責務

ちない．わざわざオブジェクト指向で設計してきたのに，ユースケースやクラス図，シーケンス図に対して実際の挙動との比較確認をしていない．中間成果物と実際の挙動を確認しなければ，ユースケースやクラス図，シーケンス図に書かれた内容と実際の動作が異なっていてもわからない，つまり，モデル～実装言語の一貫性が保証できないわけである．これは，再利用による開発の効率化をめざしたオブジェクト指向として，あってはならない姿である．

なぜ中間成果物による確認をしないのか？

- デバッグ環境と中間成果物を作成する作業環境が場所的に離れているため確認をおこたる
- 納期が迫っているため，手順を知っているのに端折った行動をしてしまい，ソースコードでデバッグしてしまう
- 旧パラダイムの開発に馴れており，ついつい“ソースコードを追いかけることが開発”というスタイルから抜けられない
- ソースコードを追いかけるのが好きである

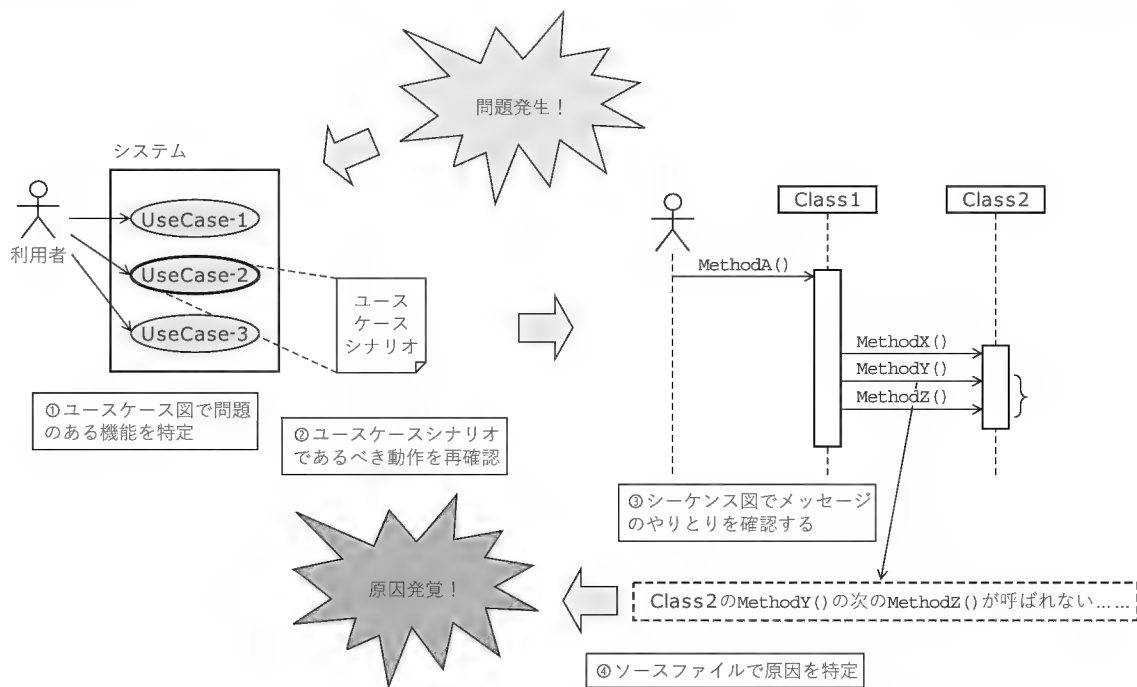
いずれの場合も，開発プロセスを定義し，その作業手順に準拠すればよい．急がば回れという事実は，経験を重ねれば重ねるだけ身にしみるものなので，開発経験を積むことが本質を理解するには近道かもしれない(**図13**)．

8.2 症状-2：作り変える，作り直す

開発担当者は，何か問題(あるいは懸念事項)が起こるとすぐに作り直すことを考える．作り直しをする際に，納期のプレッシャから，なぜそのように作り直すかが十分にレビュー(議論)されないまま作り直してしまうので，いつまでたってもモデルが落ち着かない．作り直しは，新規開発と同様に開発工数がかかるため，開発の効率化に正面からたてつく形になってしまう．はたしてこれはオブジェクト指向といえるのだろうか？

なぜ作り変えてしまうのか，作り直してしまうのか？

- オブジェクト指向は作り変えを減らすことで生産性を上げるという本来の目的を，忘れているかまたは理解できていない
- 人の作ったモデルやプログラムが気に入らない

〔図13〕中間成果物を利用してデバッグする

- 自分で作らないと気がすまない．プログラムを書くことが好き，モデルを一から作成することが好き

たとえ他人が作ったモデルでも，理解できれば再利用ができる．ソフトウェア開発の生産性を向上させるための手段として，オブジェクト指向を採用しているということをしっかり理解して，開発に挑むべきである．効率的に開発できれば余裕ができるので，できた余裕で他の部分のモデル化を行うなり，既存のモデルを改善するなりの作業を進められる(**図14**).

8.3 症状-3：作らない

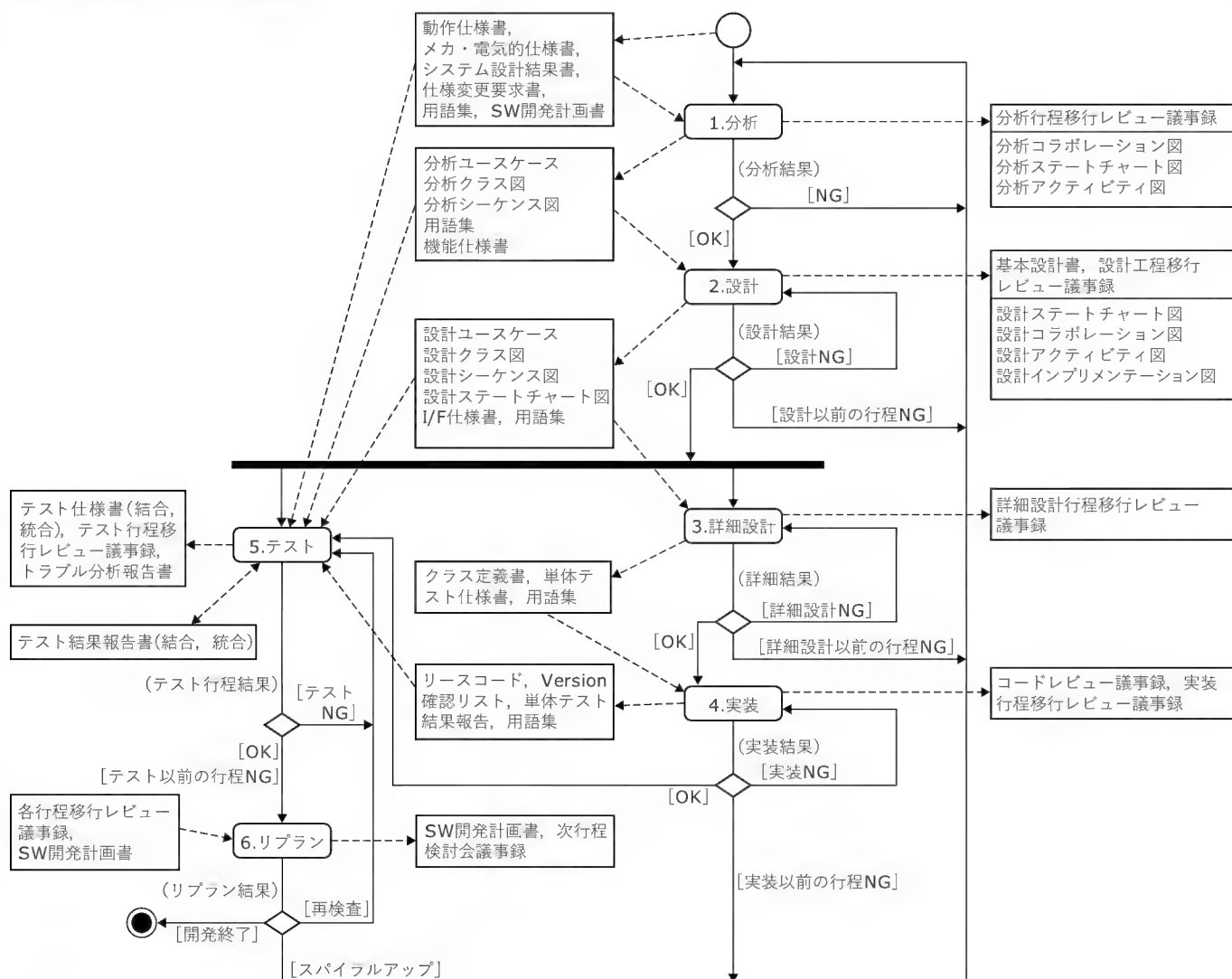
開発プロセスの各ステージで，開発プロセス定義に中間成果物として作成が規定されているもの，たとえば，ユースケース図，クラス図，シーケンス図のうち，開発後期あるいはウォータフォール型の開発プロセスの後期に重要な役割をもつユースケースやシーケンス図を作成しない傾向がある．とくにCMM-Level 1の組織でオブジェクト指向を適用すると，このようになってしまう傾向がある．よって，最低でもオブジェクト指向技術を採用する組織にはCMM-Level 2以上の開発管理能力が必要である．場当たり的開発をしている組織では，モデル間の一貫性の管理ができないのはあたりまえである．

なぜ中間成果物を作らないのか？

- 作り方がわからないか，慣れていないので作成に時間がかかる
- 必要なことは理解できているが，納期を優先しなければいけない状況であり，作成をおこたってしまう
- 中間成果物がソフトウェア品質を得るための必要条件であることを理解できていない

中間成果物は，必要だから作成するわけで，十分条件ではない．このことを開発プロセスに定義し，中間成果物による確認がなければ開発できているといえないというロジックを開発者全員で共有するべきである(**図15**).

〔図14〕簡単に再利用できれば，開発効率がアップする

〔図15〕開発プロセスと中間成果物

9 何をもってオブジェクト指向というのか

ソフト開発担当者は,「オブジェクト指向とは何か」,「オブジェクト指向をどう利用しようとしているのか」を考えずに, または議論せずに, 上司やリーダーの業務指示により受動的に開発する傾向がある. オブジェクト指向は哲学的な要素をもっているので, 開発を始める前に, この開発方法論に関しての議論をおこたると, オブジェクト指向の提供するソリューションと開発の現場にある問題の関係がつかず, 問題に対するソリューションが成立しない.

したがって, 開発開始時や開発中にオブジェクト指向をどのようにとらえ, どのように利用しようとしているかについて, 推進者と開発担当者が十分に議論し, 方針や方向性, 注意点や管理項目を共有できるような環境(チーム編成や組織体制)が必要である. これらの情報を共有できることで, 推進者の意図したとおりの開発が末端の担当者にまで浸透することになる. また, 開発中には初期の方針が忘れられる傾向があるが, 開発プロセスの中に, 共有すべき情報とチェックポイントを明確にしておけば, 開発中に起こる方針からの逸脱や脱線が未然に防止できる.

なぜオブジェクト指向で効率的に開発を進められないのか?

- 開発担当者がオブジェクト指向を選択した意図や背景を理解していない.
- オブジェクト指向で何をねらうのか, その方針が明確になっていないので, 開発中のモデルが冗長になってしまう
- 開発中の方針変更で, モデルの見直しが必要になる

基本的には,「問題は何か」ということを明らかにしたうえで,「オブジェクト指向を採用して問題を解決する」というシナリオを明らかにしておくべきである. モデルが冗長になる場合は, 分析モデルでコアとなるモデルをおさえ, 設計モデルでの冗長度をなるべく小さくする工夫が必要である.

すぎうら・ひでき 富士ゼロックス(株)

第6章

技術面，社会面，経済面，政治/法律面からの考察

事業としての組み込み機器開発の課題を整理する

井上　樹/川口　晃/佐藤啓太/橋本隆成

組み込みシステムの開発を効率化するのに何が有効かというテーマに対し，特集第１章から第５章にかけては，現場のエンジニアの立場から解説した．本章から第９章にかけては，視点を変え，開発をサポートするコンサルタントの立場から解説する．この第６章では，組み込み機器開発の現場が取り組むべき課題の整理，体系化を行う．
（編集部）

本章から第9章までは，多くの会社や社内の事業部に対して組み込みソフトウェアに関するコンサルティングを行っているコンサルタントの視点から，事業（ビジネス）としての組み込みソフトウェアを考える．

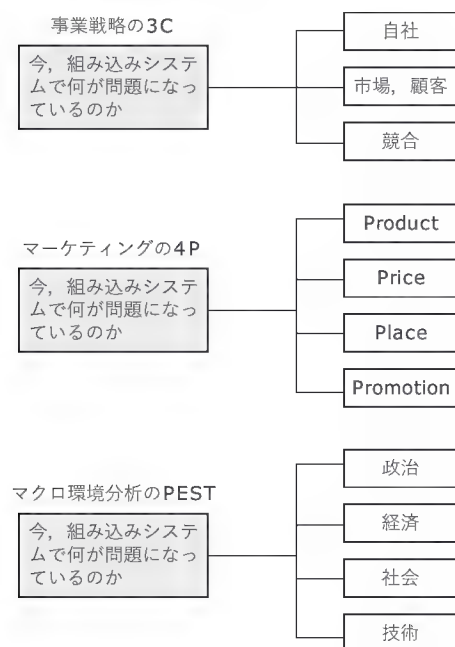
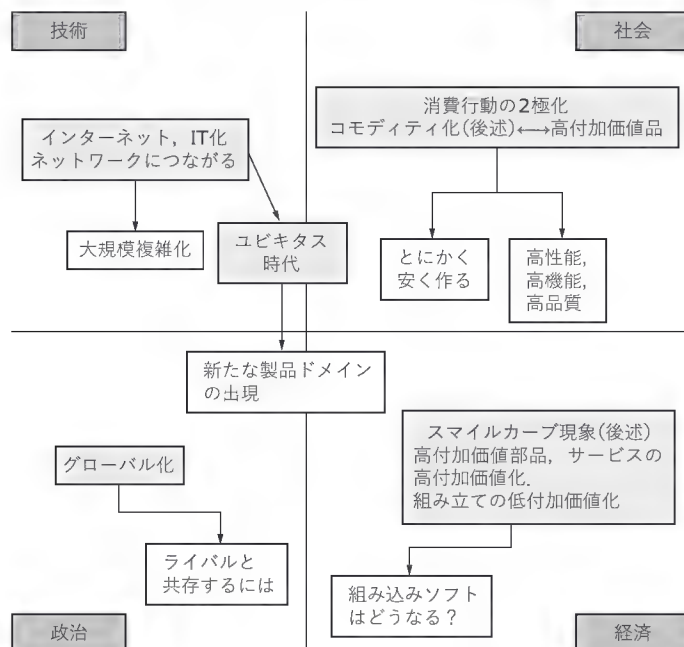
まず本章で，現状の組み込みソフトウェアが取り組むべき課題を整理し体系化する．第7章以降では，技術に関する課題，組織・ビジネスに関する課題，人的な課題について，取り組むべきポイントを述べる．

コンサルタントは，実際の事業から一歩退いた立場にあることもあり，より客観的に事業を評価できる面をもっている．このコンサルタントの目で見た現在の組み込みソフトウェア事業の課題を，分析/整理する．

分析の視点としては，いろいろな枠組みがある．たとえば，**図1**に示すとおり，事業戦略の分析に用いる「3C」（自社，市場・顧客，競合），マーケティングの「4P」（製品，価格，チャネル，プロモーション），マクロな環境分析に用いる「PEST」（政治，経済，社会，技術）などである[1]．

ここでは，環境分析のフレームであるPESTの視点で，事業としての組み込みソフトウェアを取り巻く環境の整理を行った．**図2**は，その結果を表したものである．さらに，政治，経済，社会，技術の各面で，組み込みソフトウェアが今後，取り組むべきテーマを抽出した．次に，そのテーマを解決するためには，どのような課題があるのかを抽出していく．

このように，多角的な視点（政治，経済，社会，技術）から，

〔図1〕事業分析の枠組み

〔図2〕事業としての組み込みソフトウェアを取り巻く環境

事業としての組み込みソフトウェアを分析することで，もれのない課題の抽出と，その体系化を行うことをねらっている．

1 技術面からの分析

1.1 大規模化

技術(テクノロジ面)でいえば，近年でもっともインパクトのある出来事は，ネットワーク接続の一般化，インターネットへの接続であろう．この出来事が発火点となって，組み込みソフトウェアにさまざまな変化が始まった．

とくに影響が大きいのは，ソフトウェアの大規模化である．それまでスタンドアロンで動いていた携帯電話，事務機器，制御機器などの組み込み機器が外部とつながることで一気に高機能化し，ソフトウェアは大規模化した．さらにユビキタスといわれるような，より高度なネットワーク利用が考えられるようになると，これまでにはなかった複合システムが現れるようになってきた．

また，組み込み機器に使用されるCPUの処理能力が飛躍的に向上した結果，これまで扱うことができなかった複雑なアルゴリズムや，ハードウェアで処理されてきた機能がソフトウェアで処理されることも多くなり，ソフトウェアの大規模化に拍車をかけることになった．

1.2 従来の組み込みソフトウェア開発

10年ほど前の組み込みソフトウェアは，数人で開発できる規模のものが一般的だった(もちろんディジタル交換機など，その当時でも大規模な組み込みソフトウェア開発例はたくさんあったが)．その数人で，要求記述から設計，実装までをこなす場合が多く，よくいえば「あうんの呼吸」，悪くいえば曖昧なインターフェースのまま，家内制手工業的なクローズドな世界で開発が行われていた． このような開発体制でも，ソフトウェアの規模が適切なうちは十分に機能し，大きな問題もなくソフトウェア開発が行われてきた．

1.3 家内制手工業の限界

しかし前述のとおり，一気に大規模化したソフトウェアを開発するためには，従来の開発体制はあまりにも脆弱で，いろいろな面でほころびが目立つようになってきた．まず，開発リソースに対して開発すべきソフトウェア量が膨大なため，ソフトウェアを外注することが一般的になった．

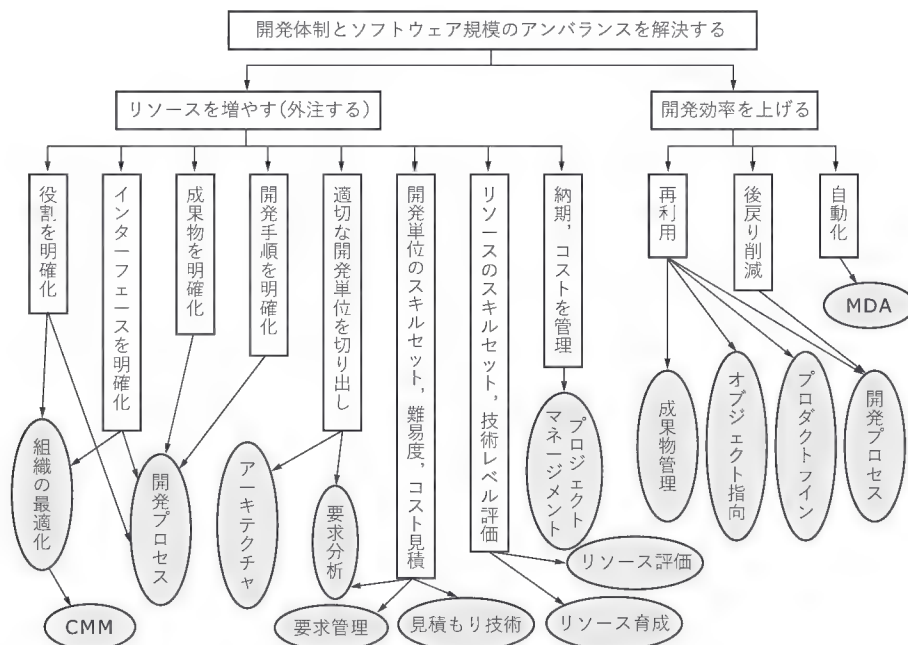
ところが，従来の家内制手工業的な開発では「あうんの呼吸」で通用したインターフェースが，外注先には通用しない．満足に要求記述を行えないため，ソフトウェア開発のほとんどすべてを外注先に丸投げせざるをえなくなる．

気がついたら自分が開発しているはずのソフトウェアの中身はほとんどブラックボックスになっている．ブラックボックスの中身を知るのは外注先だけで，何か問題が生じた場合，自社では対処のしようがなく，できるのは外注先に電話するだけといったことがごく一般的に起こるようになっていった．

1.4 技術面でのテーマ

このように，現在，組み込みソフトウェアが抱える問題の多くは，開発体制が想定していたソフトウェア規模と，実際に開発しなければならないソフトウェア規模の不一致に端を発していると考えられる．そこで，「開発体制とソフトウェア規模のアンバランスの解決」をテーマに，技術面での課題を抽出してみた．**図3**にその結果を示す．

これらの課題は，以降で述べる社会面，経済面，政治/法律面

〔**図3**〕**技術面からの分析**

それぞれの課題と合わせて，本章の最後で整理，体系化を行う．

2 社会面からの分析

2.1 消費行動の2極化

近年，消費行動の2極化が指摘されている[2]．日常品などどうしても必要なもの(コモディティ化)はできるだけ安く(必需的消費)，ブランド品などの高付加価値なものには出費を惜しまない(選択的消費)傾向にあるという．

2.2 消費行動の2極化が組み込みソフトウェアに与える影響

消費行動の2極化が組み込みソフトウェアに与える影響を分析してみる．まず，コモディティ化する商品に関しては，とにかく安く作ることが求められる．それに対して，高付加価値製品の場合は，多少開発費がかかったとしても，高機能，高性能，高品質が求められる．同じ技術を用いて作った組み込み機器であっても，コモディティ化する製品と，ブランド品のような高付加価値製品では，投資と回収の考え方がまったく異なってしまう．つまり，製品企画や開発段階でマーケットでのその製品の位置付けなどを十分考慮した，よりマーケットオリエンテッドな開発スタイルが求められている．

2.3 社会面でのテーマ

そこで，社会面からの分析テーマを，「マーケットオリエンテッドな開発の必要性」として課題の抽出を行ってみた．**図4**にその結果を示す．

2.4 コモディティ化する製品の開発スタイル

コモディティ化する製品は，薄利多売で開発コストを回収する必要がある．しかし，後述するように組み込み機器の製品ライフサイクルは短くなる一方である．そこで，同一製品系列で何世代にもわたって回収する，あるいは，より広い範囲の製品系列で回収するなどの方策が求められる．これは，より長期間にわたって再利用できるソフトウェア，あるいは，より広い範囲に再利用できるソフトウェアなど，再利用を強く意識した開発スタイルが求められていることを意味する．

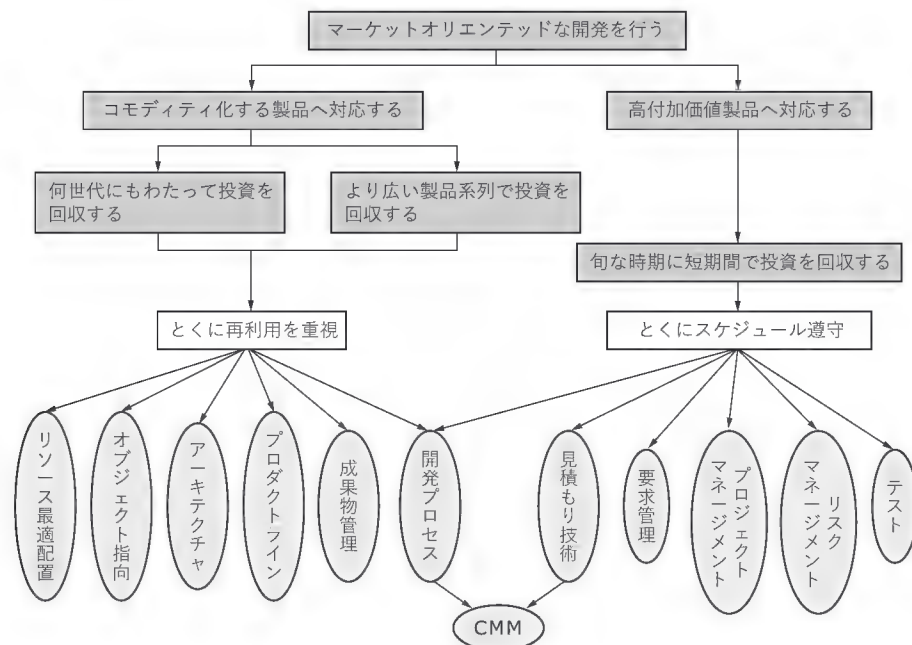
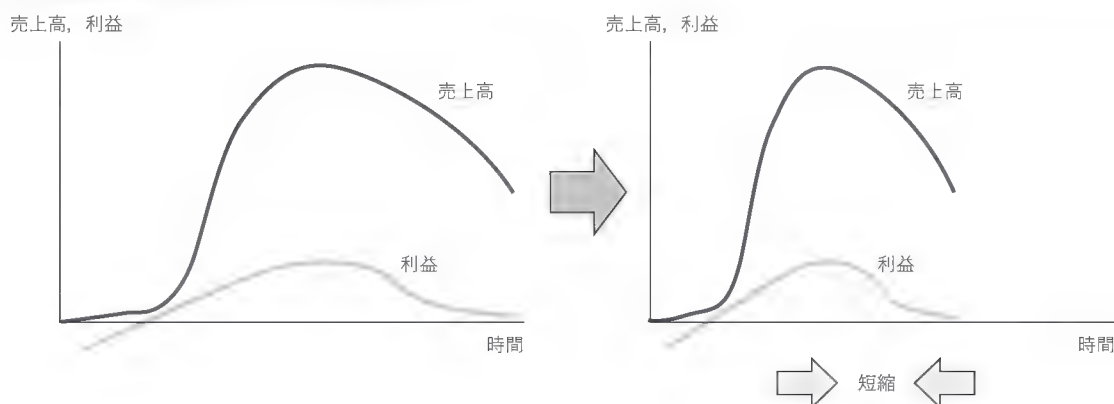
2.5 コモディティ化する製品を支える基盤

このような再利用を計画的に行うためには，製品ロードマップを早い時期から具体的に定め，その意図を実現できるソフトウェアアーキテクチャを構築し，計画どおりに再利用部品を開発することが求められる．製品ロードマップを作成する企画力，その企画から将来にわたって使用可能なソフトウェアアーキテクチャを構築する要求分析能力，さらにそのアーキテクチャ上で動作する再利用可能なコンポーネントなどの開発能力が求められる．つまり，コモディティ化する組み込み機器の開発には，たとえばプロダクトラインやオブジェクト指向技術に代表される基盤が整っていることが必要になる．

2.6 高付加価値製品の開発スタイル

一方，高付加価値製品の場合は，いわゆる「旬な時期」にコスト回収を行うことがとくに重要である．近年，組み込み機器の製品ライフサイクルが太く短くなる傾向があり，高付加価値製品に見合う開発コストを回収できる時期は長くない[3]．

図5に，組み込み機器における製品ライフサイクルの変化を示す．この時期にタイミング良く製品を販売できないと，みすみす開発コスト回収の機会を損失することになる．製品の販売スケジュールを遵守し，予定されたタイミングで製品を投入す

〔図4〕社会面からの分析

〔図5〕組み込み機器における製品ライフサイクルの変化

ることが非常にたいせつになる．コストや品質をクリアしたうえで開発スケジュールを遵守することが，高付加価値製品の開発スタイルといえる．

2.7　高付加価値製品を支える基盤

コスト，品質などを満足したうえで，製品化スケジュールを遵守するためには，開発プロセス，プロジェクトマネジメント，リスクマネジメント，テスト技術などに加えて，要求ごとに過去の開発実績を蓄積し，その情報から正確に開発期間を見積もり，組織として開発プロセスを共有・管理し，安定的な開発を可能にするなどのCMMに基づく基盤が必要になる．

3　経済面からの分析

3.1　スマイルカーブ現象と組み込みソフトウェア事業

読者の皆さんは，スマイルカーブ現象という言葉をご存じかもしれない．**図6**に，一般的にいわれるスマイルカーブを示す．製造業についていわれることが多いが，コアな部品とサービスに付加価値が集中し，その真ん中に位置する組み立ての付加価値が低くなる現象を指す．この現象を組み込みソフトウェアに当てはめるとどうなるだろうか？

図7は，情報化投資に当てはめた場合のスマイルカーブである[4]．企画や運用保守の付加価値が高く，開発は低くなっている．組み込みシステムは，どちらかといえば，こちらに近いのだろうか？

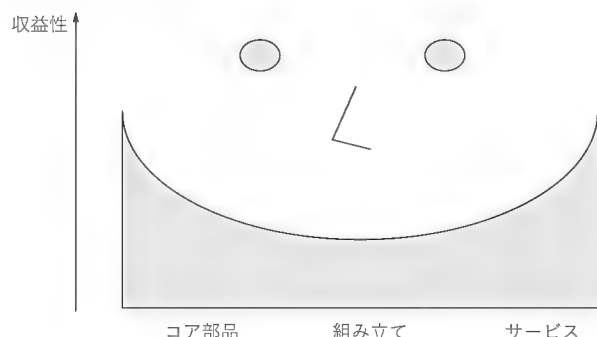
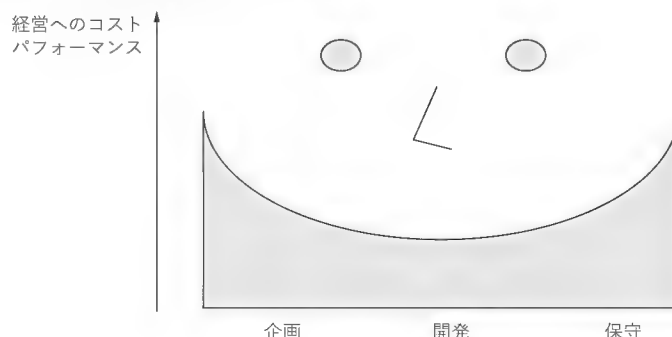
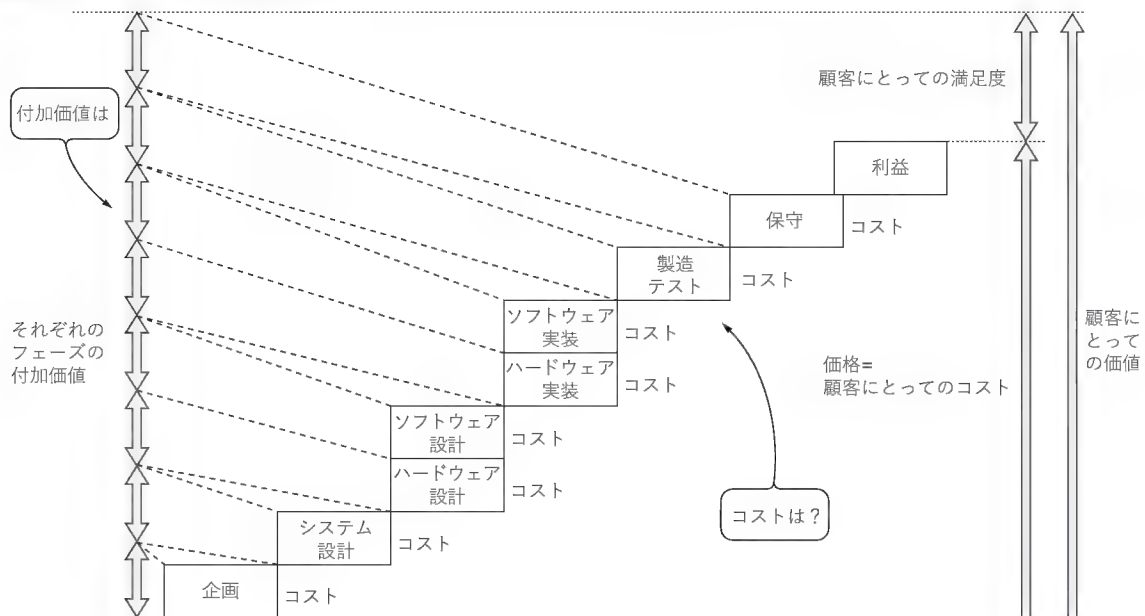
3.2　組み込みソフトウェア事業における付加価値とは

そもそも，この議論を行うためには，組み込みソフトウェア事業におけるコストと付加価値とは何かを正確に議論することが重要だと思う．よくいわれることであるが，組み込みソフトウェアはハードウェアの付属品的な扱いだった時期がある．その名残りからか，組み込みソフトウェアの事業としてのコストと付加価値について議論がつくされていないように思う．

この議論をしっかり行わないと，前述のスマイルカーブ現象が組み込みソフトウェアにも当てはまるのか，当てはまるとすれば，ビジネスシステムにおいてどの部分の付加価値が高くなり，どの部分が低迷するのかも議論できない．

図8に，組み込みシステム機器におけるビジネスシステムの例を示す[5]．この図では，すべての開発フェーズで，同じだけのコストと同じ付加価値が生み出されることになっている．実際にはどうだろうか？

たとえば，組み込みソフトウェア設計のフェーズは，全体に対するコストの割合はいくらで，全体の何%の付加価値を生んでいるのか？ もし，コストの割合が高いにもかかわらず，他のフェーズより付加価値が低い場合，そのフェーズをどのように改善すれば良いのかを検討する必要がある．さらに，自社内で

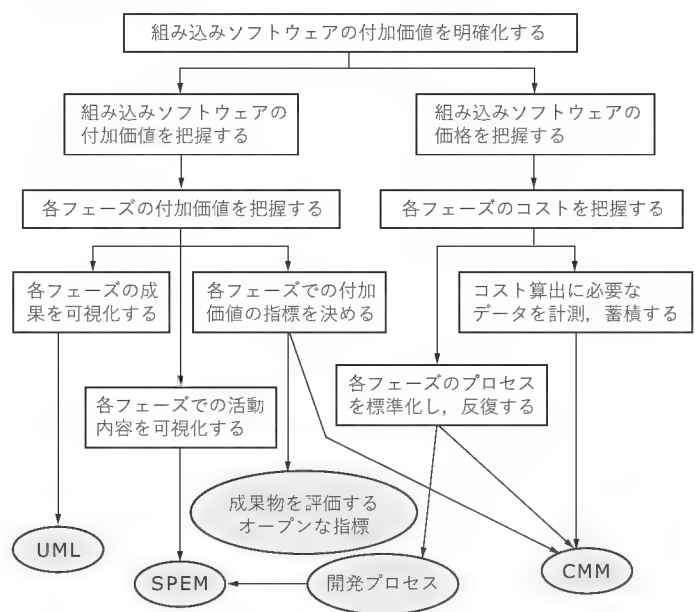
〔図6〕一般的なスマイルカーブ

〔図7〕情報化投資のスマイルカーブ

〔図8〕組み込み機器のビジネスシステムと付加価値

そのフェーズを実施する是非も検討しなければならない.

また，コストの割合が低いにもかかわらず，大きな付加価値を生むフェーズについては，自社内だけで占有せず外部へ販売する戦略も検討できる．このような議論を精密に行わないと，事業としての組み込みソフトウェアの評価をまっとうに行うことができない.

3.3　経済面のテーマ

そこで，経済面でのテーマを，「組み込みソフトウェアの付加価値を明確化」として，取り組むべき課題の抽出を行った．結果を**図9**に示す.

この中で，各開発フェーズでの付加価値の指標として，成果物を評価するオープンな指標を課題としてあげている．これは，各フェーズの具体的な成果物(たとえば，分析モデルや設計モデル，ソースコード)を誰もが公平に評価できるような基準のことを指している．組織の成熟度に関してはCMMがあるが，成果物にもこのような指標が必要ではないだろうか．なお，SPEMとは，Software Process Engineering Metamodelの略である[6].

〔図9〕経済面の分析

4　政治，法律面からの分析

4.1　グローバル化

東西冷戦構造の終結で始まったグローバル化の波は，非常な勢いで世界中を覆いつつある．日本の組み込みソフトウェア事業についても例外ではない.

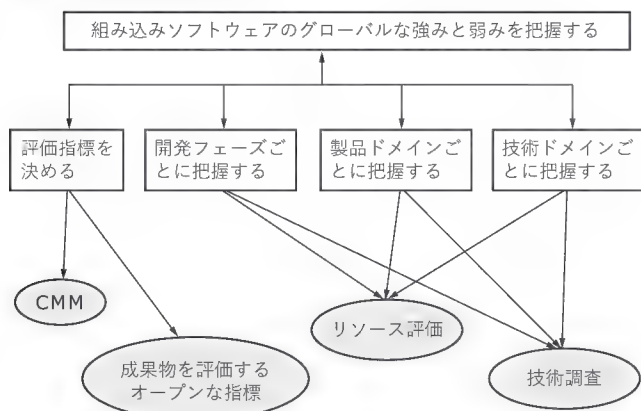
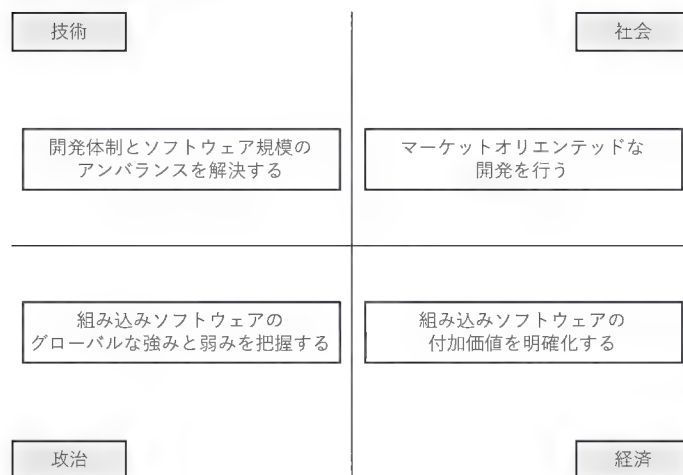
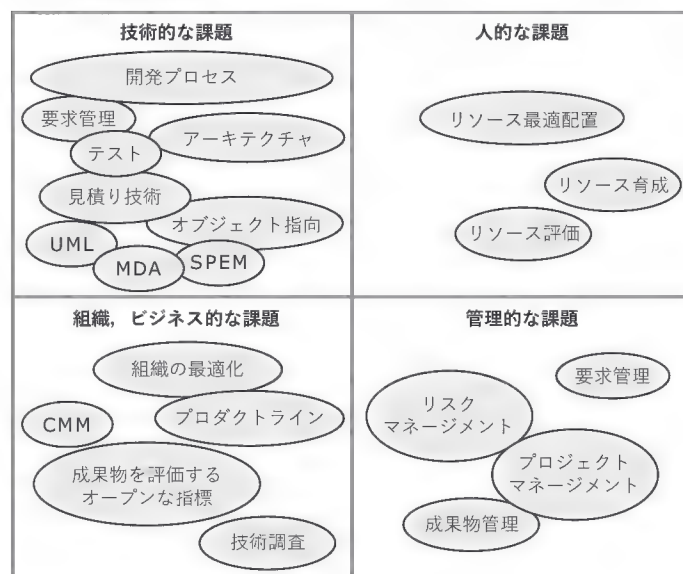
4.2　日本の組み込みソフトウェアの実力は？

一般に日本の組み込みソフトウェア開発力は世界レベルの競争力があるといわれている．ところが，実際にどの部分が強いのか，どうして強いのか，将来にわたってその競争力を維持できるのかというような具体的な話になると，なかなか説得力のある意見が出てこない.

たしかにある時期には，商品としての組み込み機器が世界中を席巻したこともあった．その時期のイメージが強いので，漠然と組み込みソフトウェアの実力も世界レベルと思い込んでいるように見えなくもない.

4.3　今，やるべきこと

今後も衰えそうもないグローバル化の勢いに負けないためには，組み込みソフトウェアのどの部分がどのようにいつまで強いのか，なぜ強いのか，弱い部分はないのか，ライバルはどこ

〔図10〕政治/法律面での分析

〔図11〕分析軸ごとのテーマ

〔図12〕課題の体系化

で，どのようにすれば勝てるのかを真剣に考えることが必要だと思う．このためには，グローバルな視点で組み込みソフトウェア事業の強み，弱みをまとめることが必要である．

4.4　政治/法律面のテーマ

政治/法律面のテーマを，「組み込みソフトウェアのグローバルな強みと弱みを把握する」として，課題の抽出を行った．結果を**図10**に示す．

強みと弱みを把握することは，なかなか簡単なことではない．もっとも早道なのは，製品開発を依頼してみることであるが，高いリスクをともなうので，実行できる企業は少ないであろう．リソース評価や技術調査などの技術，方策を早急に確立することが必要だと思う．

おわりに

以上，政治，経済，社会，技術面から，組み込みソフトウェアの課題の抽出を試みた．**図11**に，課題を抽出する際に用いたテーマをまとめておく．また，抽出された結果をまとめたものを**図12**に示した．

おおまかに分類すると，技術的課題，組織・ビジネス的な課題，人的な課題，管理的な課題に大別することができそうだ．いかがだろうか．この課題は含まれていないが？など，いろいろな意見が出そうな結果となった．すべての課題が網羅されているかどうか，ご意見を頂戴したいところである．

筆者らとしては，組み込みソフトウェアの付加価値を明確化すること，ならびに，グローバルな視点で強みと弱みを把握することが今後の組み込みソフトウェア事業を考えるうえで重要だと思う．その意味では，成果物を評価するオープンな指標の確立，技術調査の推進，リソース評価方法の確立，CMMの活用などが必要になると感じている．

参考文献

1）グロービス・マネジメント・インスティチュート著，『MBAクリティカルシンキング』，ダイヤモンド社
2）第一生命経済研究所，「住宅ローンと家計動向（個人消費編）」，『日本経済の羅針盤』 http://group.dai-ichi-life.co.jp/dlri/rashinban/pdf/et01_40.pdf
3）経済産業省 関東経済産業局，『製品の電子化と産業集積地の産業構造変化に係る実態調査報告書』 http://www.kanto.meti.go.jp/tokei/
4）井上 実，「情報化投資のスマイルカーブ」 http://homepage2.nifty.com/minoru-inoue/IT-smile.htm
5）齋藤嘉則，問題発見プロフェッショナル「構想力と分析力」，ダイヤモンド社．
6）Software Process Engineering Metamodel version 1.0， Object Management Group, Inc. http://www.omg.org/technology/documents/formal/spem.htm

いのうえ・たつき　(株)豆蔵
かわぐち・あきら　(株)ガイア・システム・ソリューション
さとう・けいた　(株)デンソー
はしもと・たかなり　ソニー(株)

第7章

組み込み系ソフト開発効率向上のための技術面の取り組みポイント

ソフトウェアプロダクトラインとプロセスの定義

井上　樹／川口　晃／佐藤啓太／橋本隆成

組み込み系ソフトウェアの開発効率を向上させるために，分析手法やオブジェクト指向の導入など，さまざまな方法がとられる．しかし現実的に，これらによってすべてが成果を得られているというわけではない．それはなぜか？　多くの要因が考えられるが，有力なものの一つとして，改善すべきポイントと導入した技術がマッチしなかった，ということがある．本章では，組み込み系のソフトウェア開発の特徴を解説し，組み込み系に見合った開発スタイルの例として，ソフトウェアプロダクトラインというアプローチを紹介する．また，ソフトウェア仕様の記述方法やモデリングツールにも言及する．（編集部）

多くの開発現場でソフトウェア開発を改善するための活動が行われている．本章ではそうした改善活動の技術的な側面を取り上げる．

ソフトウェア開発改善の技術的側面として，新規技術の導入や開発手順の見直しといったものが挙げられる．システム分析といった上流工程に関する技法や，オブジェクト指向を用いたソフトウェア開発といった技術を導入したという覚えのある方も多いだろう．しかし，そうした技術の導入が実を結ばないことも多い．その原因としてさまざまな要因が考えられるが，その一つとして，改善すべきポイントに対して導入した技術が適切でないということが考えられる．たとえていえば，改札の数が少ないためにホームに人があふれている駅で，ホームに人が多いからという理由で，改札への階段を増設してしまうようなものである．

そこで，導入する技術が的を外さないよう，多くの組み込み系ソフトウェア開発の改善の際にポイントとなるものと，そこで利用可能なソリューションについて，以下に述べる．

1　組み込み系ソフトウェア開発に適した技術の導入

1.1　組み込み系ソフトウェア開発と業務系ソフトウェア開発

ソフトウェア開発を改善するために導入した技術が効果を発揮しない理由の一つとして，組み込み系ソフトウェアの開発と業務系ソフトウェアの開発の開発スタイルがまったく異なるという点が挙げられる．

非常に乱暴な分類だが，ソフトウェアを大きく二つに分けると，組み込み系ソフトウェアと業務系ソフトウェアとに分けることができる．組み込み系ソフトウェアとは，おもに機械制御が中心となるようなものであり，業務系ソフトウェアとは，企業の基幹業務などで用いられるようなデータ処理を中心としたものである．

両者を比較した場合，業務系ソフトウェア開発は組み込み系ソフトウェア開発に比べて開発に関する考え方が開放的であり，組織や分野を超えて多くの研究者やコンサルタントが関わっている．

また，ソフトウェアの大規模化/複雑化が早くから問題となっており，組み込み系ソフトウェア開発よりも早い段階から上流工程のソリューションやソフトウェア工学に関する研究・技術が創出され，整備，展開されている．

一方，組み込み系ソフトウェアは，業務系ソフトウェアと比較すると，ソフトウェア開発に関する組織や分野を超えた意見交換が行われるようになってからまだ日が浅い．また，組み込み系ソフトウェアでは，そのソフトウェアの実行環境の制約の厳しさから，リアルタイム性や品質といったテーマを中心とした下流工程が重視されてきた．

そうした組み込み系ソフトウェアの開発だが，最近になって，他社との差別化を図るために上流工程を重視するようになったり，大規模複雑化するソフトウェア開発への対応が必要となってきている．しかし，前述したような組み込みソフトウェア開発のそれまでの状況ゆえに，上流工程を研究していた研究者やコンサルタントは非常に少なく，この部分のソリューションは圧倒的に不足している．そのため，業務系ソフトウェア開発向けのソリューションが組み込み系ソフトウェア開発のためのソリューションとして用いられている．

しかし，業務系ソフトウェア開発と組み込み系ソフトウェア開発は元来大きく異なるものである．そのため，導入する技術は，その違いや特徴（**表1**．ただし，すべての組み込み系ソフトウェア開発に当てはまるというわけではない）を認識し，組み込み系ソフトウェア開発のためのソリューションとして各開発組織の事情にあったものに仕立てなければ，ソリューションとしては的を外れたものとなってしまう．

たとえば，業務系ソフトウェアのように十分なCPUパワーとメモリ量を前提としたような技術を導入しても，組み込み系ソ

フトウェアの開発には向かないのである．

以下，簡単に業務系ソフトウェアと組み込み系ソフトウェアの違いを比較する(**表1**，**図1**)[注1]．

また，**表1**の内容と一部繰り返しになるが，特記すべき組み込み系ソフトウェア開発の特徴として以下のものがある．

▶シリーズ開発が中心

組み込み系ソフトウェアは，基本的にシリーズ開発が中心である．つまり，最初に製品シリーズの計画があり，その計画に基づいて製品のロードマップが引かれる．製品はそのロードマップにしたがい，製品バリエーションの開発，機能追加が行われていく．つまり，類似製品が同時に作られ，その後，その機能を引き継いだ次の製品が作られるというスタイルである．

▶再構築が中心

組み込み系ソフトウェア開発では，ソフトウェアを一からすべて作成するということはあまりない．多くの場合，既存資産を活用しながらソフトウェアを構築していくことになる．そのため，オブジェクト指向やUMLを利用して新規に開発を行おうということになると，既存資産を作り直すことになり，膨大な時間とコストをかけて保証してきた品質を，また最初から検証しなければならない．

世の多くのソフトウェア開発方法論は新規開発を前提としており，こういった再構築中心のソフトウェア開発に関する方法論はほとんど存在しない．

▶複合システム開発

組み込み系ソフトウェアでは一つの製品を構成するために複数のシステムが組み合わさっていることが多い．個々のシステムを見てみるとそれだけで完成したシステムなのだが，複数のシステムが協調しあうことでより大きなシステムを構成している．

また，最近では既存システムを組み合わせてより大きなスーパーシステムを構築することによって，競争力のある製品を作り出そうという開発もある．現状ではそうした，ある製品を複数のシステムに分割していくトップダウン型のシステム分割や，既存システムを組み合わせてスーパーシステムを構築していくボトムアップ型のシステム統合を考慮した開発方法論は存在していない．

こうした特徴をもつ組み込み系ソフトウェア開発に対して，現状ではこれらの特徴をふまえたような改善活動はほとんど行われていない．以下では，そうした組み込み系ソフトウェア開発に見合った開発スタイルの例として，**ソフトウェアプロダクトライン**(**Software Product Line**)というアプローチを紹介する．

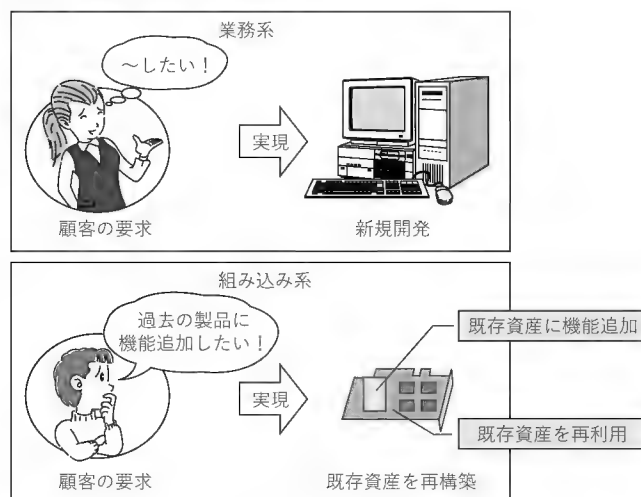
1.2 ソフトウェアプロダクトライン

前述したように，組み込み系ソフトウェア開発の特徴の一つとして，開発のほとんどがシリーズ開発であることが挙げられる．つまり，多くのプロジェクトが，それまでに存在していた製品をベースにした後継製品の開発プロジェクトであるということである．そのため，すべてを一から開発する製品はほとんど存在しない．

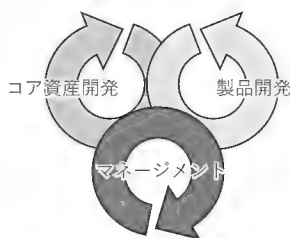
すべてを一から開発することがほとんどないのは，機能がほとんど同じだからということだけが理由ではない．機能が同じでも，新しい手法や技術で同じ仕様のソフトを作り直すということを行わないのは，すでに存在しているソフトウェアは品質が十分に保証されたものだからである．

〔表1〕業務系と組み込み系の比較

	業務系ソフトウェア	組み込み系ソフトウェア
発注	多くが一回だけの開発で終了する． 開発後は保守	定期的，非定期的に改良を加えながら繰り返して開発． 構成の違いなどによりバリエーションを開発
ソフトウェア開発	ソフトウェア開発＝システム開発と考えられる	ソフトウェア開発は製品開発，システム開発の一工程
ソフトウェア要求	システム利用者によって決められる	過去の機種の持つ機能，市場動向，要素技術などによって決められる
再利用	同一の内容を持つソフトウェアを開発することは少ないので，再利用は困難	以前に開発したソフトウェア資産を活用しながら開発を行なうのが普通
開発の始まり方	システムへの要求がそろっていることを前提として開発を始める	システムへの要求が決まりきらないうちに開発が始まる
開発環境	上流から下流まで，充実している	下流工程は充実しているが，上流工程については弱い． 開発環境自体を自主開発する場合もある
リソース(CPU，メモリ，HDDなど)	リソースは変更可能なため，十分なリソース量を確保できる	リソースを変更することは基本的に不可能なため，リソース量には制限がある

〔図1〕業務系と組み込み系の違い

注1：ここで対象としている業務系ソフトウェアは，おもに受託開発の場合である．パッケージソフトウェアの開発の場合，あてはまらないこともある．

〔図2〕プロダクトライン開発を構成する三つの要素

〔図3〕ドメインエンジニアリングとアプリケーションエンジニアリング

新たに作り直すということは，その部分の品質をもう一度作り込まなければならないということである．こうした考え方が，業務系ソフトウェア開発と組み込み系ソフトウェア開発が大きく異なる部分である．

以上のように，過去の資産を活用しながら新しい製品を作っていくというやり方が中心となる組み込み系ソフトウェア開発だが，どのように過去の資産を活かすのかということに関しては，体系立てられた開発手順やマネージメントが存在している組織はほとんどない．

通常，過去の資産を活かすといっても，それまでの開発で作られた成果物をすべてコピーし，それらから使えそうな部分を技術者が経験と勘で切り出し，それらを組み合わせ，修正，追加して新製品が作られる．しかし，近年の大規模化する組み込み系ソフトウェアに対し，勘と経験に頼った場当たり的な過去の資産の活用は限界に来ている．

そうした，シリーズ開発と混沌とした再利用体制が特徴となる組み込み系ソフトウェア開発に合うようなソフトウェア開発のアプローチとして，ソフトウェアプロダクトラインがある．これはCMM(ソフトウェア能力成熟度モデル)の発信元でもあるカーネギーメロン大学のソフトウェア工学研究所で考えられたものである．

ソフトウェアプロダクトラインの特徴は，システム開発に関わる活動を，コア資産の開発(Core Asset Development)，製品開発(Product Development)，マネージメントと大きく三つに分けていることである(**図2**)．

コア資産開発とは，プロダクトラインスコープの決定，コア資産の開発，製品開発計画の三つの活動を行うことで，製品開発を行うために基盤を作る活動である．プロダクトラインスコープの決定では，どういった製品シリーズを対象としてプロダクトラインを考えるのかということを決定する．

コア資産の開発では，プロダクトラインスコープに含まれる製品群を開発するために必要となる部品(コア資産)の開発を行う．製品開発計画ではプロダクトラインスコープに含まれる製品群をどのようなスケジュールで開発していくのかということを決定する．

また，製品開発とは，コア資産開発で作成された計画とコア資産に基づいて，製品を開発する活動である．また，製品開発のためには計画とコア資産以外に，開発対象となるそれぞれの製品への要求が必要となる．

最後にマネージメントとは，コア資産開発と製品開発両者の人員配置，調整，監視を行う活動である．

ソフトウェアプロダクトラインでは，コア資産開発とマネージメントをあわせた活動をドメインエンジニアリング，製品開発とマネージメントをあわせた活動をアプリケーションエンジニアリングと呼んでいる(**図3**)．

具体例でソフトウェアプロダクトラインの活動を考えると，たとえば，情報通信機器というスコープに対してソフトウェアプロダクトラインを適用すると，通信用プロトコルスタックの開発といった部分がドメインエンジニアリングに該当する．一方，そういったプロトコルスタックのようなコア資産を組み合わせて，携帯電話やPDAを開発するのが製品開発という活動に該当する．

ソフトウェアプロダクトラインでは，以上のような活動を通じて製品が開発されていくが，その中で行われる製品ロードマップを作成し，シリーズのコア資産とその開発計画を立て，計画的にコア資産の開発と再利用を行っていくという方式は，シリーズ開発中心の組み込み系ソフトウェア開発に非常に向いている方法論であるといえる．

ここで紹介したソフトウェアプロダクトラインは一例でしかないが，組み込み系ソフトウェア開発の特徴を考え，それにより合致した技術を利用することで，的外れな技術を導入することによる改善活動の失敗を回避することができるのではないだろうか？

2 現状の整理・洗練・体系化

2.1 プロセスを定義するということ

組み込み系ソフトウェア開発改善の次のキーポイントとしては，それぞれの開発組織で現在行われているソフトウェア開発の手順や技術を整理・体系化するという点が挙げられる．いい方を変えると，その組織の標準的なプロセスを定義するということである．

現在，多くの組み込み系システム開発では，その組織で標準となるプロセスが整備されていない．そのため，次のような問題が発生してしまっている．

▶ソフトウェア開発を計画的に進められない

ソフトウェア開発に際し，開発手順が整理・体系化されていないために，その開発手順や判断基準，作成する成果物といったものが組織や人によってばらばらになっている．つまり，場当たり的にソフトウェア開発が進んでしまうという状況である．そのため，完成したソフトウェアの品質がそろわなかったり，開発期間の見積もりの精度が低かったりといった問題が発生する．そうした状況は「作ってみなければわからない」ということであり，これではソフトウェア開発を計画的に進めることはできない．つまり，ソフトウェア開発をマネージメントすることができないという状況を招くことになる．

▶仕様と実装との断絶

プロセスが定義されていない現状の開発を見ると，要求仕様と実装との間をつなぐもの（設計資料・モデル）が少ない，もしくは資料の内容が実態と異なっているという状況がよくある．極端ないい方をすると，プロセスがないということは中間成果物がいっさい役に立たないため，最終成果物であるソースコードのみが信用できるドキュメントとなってしまう[注2]．そのため，仕様の追加や変更を，実装のどの部分に反映させればよいのかがすぐにわからないという問題が発生する．

また，そういった状況では，残っている成果物がソースコードのみで，ほかの仕様に相当する資料は実装を担当した者の頭の中にのみ存在するという場合が多く，実装をほかの製品向けに再利用することを難しくしている．

▶実装の一枚岩化

プロセスが定義されておらず，ソースコードがもっとも信頼できるドキュメントであるという状況の場合，ソフトウェア全体を見渡すことのできる資料がないこともある．そうした場合，ソフトウェア全体の構造に対する十分な検討が行われておらず，ソフトウェアのモジュール化がうまくできていないことが多い．

そのような場合，実装の追加や修正・変更が，ほかの部分のどこまで影響してくるのかわからず，恐くて実装に触れられないという状況を生む．その解決策として，元のシステム全体をブラックボックスとして扱い，上から皮を被せる形で差分を実装するという方法が取られる．

その結果，ソフトウェアの一枚岩化が進むこととなる．一枚岩化されたソフトウェアは効率的な再利用はできず，冗長なコードを含むことによるソフトウェアの肥大化，低品質化をまねくことになる．これは，組み込み系ソフトウェアの性質とはまったく反対方向のソフトウェアとなってしまう．

▶明文化されない「仕様」の存在

前述した，仕様と実装の断絶とも関連するが，仕様が明確に記述されていなかったり，場当たり的に非機能要求に対応していたりといった状況であると，明文化されない仕様が実装されることがある．

たとえば，速度やリソースなどの非機能要求の実現や，実装開始後にわかったデバイスの制約などを，実装上の華麗なテクニックや，過剰（とその時点では思える）仕様の切り捨て，あるいはフラグや条件文の追加で解決するといったところから，仕様書にはない「仕様」が実装される．そうした，実装時に追加された仕様は，それが含まれたコードを引き継いだ開発者にとっては，トラップが仕掛けられているようなものである．そのようなソースコードに対して，ソフトウェアの品質を上げるのは難しい作業となる．

上記はそれぞれが独立した問題ではない．現状の開発の進め方や開発を行う組織体制の抱える問題点と，そこから派生する開発者へのさまざまなプレッシャといったバックグラウンドが，上記の諸問題を発生させている遠因となっている（**図4**）．これは，技術の領域だけの問題ではなく，他章で述べている組織や人の問題でもある．

こうした，場当たり的，マネージメント不在，ソースコードしか信用できない，必要な情報が開発者の頭の中にしかない，といった状況から脱却するには，現状を整理，体系化したプロセスの定義が必要である．

プロセスの定義をするには，まず自身がどのような開発を行っているのかを把握する必要がある．そのためには，現在行っている開発手順の文書化と，アセスメントを実施するのが良い．アセスメントとは，組織内のソフトウェア開発がどのように行われているのかを調査し，何ができていて，何ができていないのかを把握する活動である．アセスメントを行うことで，プロセスの現状と，その後のプロセス定義の方向性を定めることができる．

以降では，プロセス定義に関する部分で特に改善のポイントとなる部分である，ソフトウェア仕様，ツールの導入，構成管理について述べる．

〔図4〕ソフトウェア開発プロセスに絡む問題

場当たり的な製品開発
仕様と実装との断絶

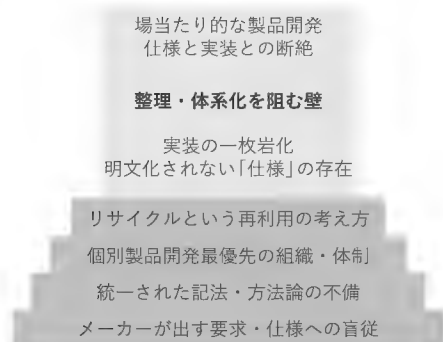
整理・体系化を阻む壁

実装の一枚岩化
明文化されない「仕様」の存在

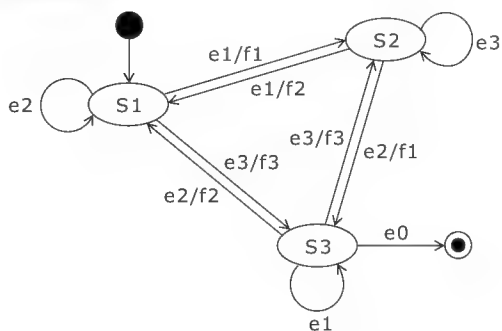
リサイクルという再利用の考え方
個別製品開発最優先の組織・体制
統一された記法・方法論の不備
メーカーが出す要求・仕様への盲従

2.2 ソフトウェア仕様

● ソフトウェア仕様がきちんと書かれていないという問題

ソフトウェア開発において，仕様が重要な位置を占める．仕

注2：ソースコードをコンパイルした結果であるバイナリに直接手を加えている場合には，ソースコードすらも役に立たない．

〔図5〕状態遷移モデル

〔図6〕アクティビティモデル

様に基づいてソフトウェアが作られるので，仕様がなければソフトウェア開発は始まらないはずである．そのため，ソフトウェア開発のプロセスの中でも仕様の定義は重要な部分である．ソフトウェアの仕様をきちんと書くところからソフトウェア開発が始まるといってよい．

そうした仕様の記述についての問題点としては，そもそもソフトウェアの仕様がきちんと作成されていないということが挙げられる．つまり，システム全体の仕様や，メカ，電気的仕様といったものは存在することが多いが，それらの間をつなぎ，システムとして整合させる役割であるソフトウェアに関しては，どのような仕様で作ればよいのかの記述がないことがほとんどなのである．

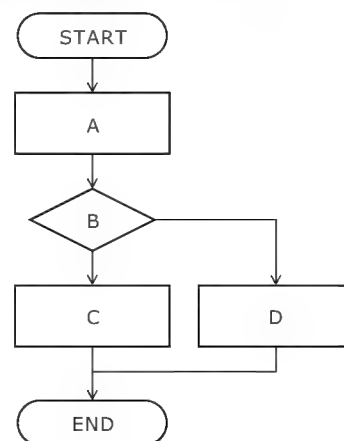
ソフトウェアの仕様といって出てくるものが，システム，メカ，電気的仕様書を束ねたものだったりするのは，よくある話である．そうした状況では，ソフトウェアの仕様を記述することは，大きな改善活動となる．また，そうしたソフトウェアの仕様を記述するにしても，意図がきちんと伝わらなかったり，曖昧な表現がされているようでは仕様としての役をなさないので，注意が必要である．

● **仕様の記述方法**

では，ソフトウェアの仕様をきちんと書くとしても，どのように書けばいいのであろうか？

たとえば，最近オブジェクト指向の導入にともなって，ソフトウェアの仕様を表すためにユースケースを用いている場合がある．ここで問題になるのが，新しい技術にとらわれてしまい，ユースケース「だけ」でソフトウェアの仕様を記述しようとしてしまうことである．

組み込み系ソフトウェアの仕様というと，ユースケースなどで記述できるような「機能」に関する仕様だけではなく，パフォーマンスなどの非機能仕様やプロトコル，ハードウェアに関する仕様，そして制御のアルゴリズムなどに関する仕様といったように，記述対象となるものにはさまざまな種類がある．

また，HMI（Human Machine Interface）を利用するようなインタラクティブなシステムや，データの一括処理といったバッチ的なもの，フィードバック系，データフロー系など，システムのタイプによっても必要な仕様の記述方法は異なってくる．

つまり，ユースケースやUMLだけですべての仕様が記述できるわけではないので，対象とするものに応じた記述方法が必要なのである．

そうした仕様を記述するためには，対象を何らかの形でモデル化することになる．以下にそうしたモデル化の代表的なものを挙げる[注3]．

▶状態遷移モデル

状態遷移に着目したモデル化である．このモデルは，仕様を記述する対象がイベントドリブンなふるまいをするもので，なおかつ状態に応じてイベントに対する反応が変わるような場合に用いるとよい．組み込み系ソフトウェアは状態機械でその全体のふるまいを表せることが多いので，ソフトウェア全体のふるまいを表したりする際に，このモデル化が行われる（**図5**）．

▶アクティビティモデル

処理のフローに着目したモデル化である．このモデルは，仕様を記述する対象が順序処理的なふるまいをする場合に用いるとよい．たとえば，ソフトウェアの提供するサービスの利用手順などは，このモデルによる表現が向いている．オブジェクト指向を使った開発でよく利用されるユースケースは，このモデルを文章で表現したものである（**図6**）．

▶構造モデル

ソフトウェアの構造に着目したモデル化である（**図7**）．このモデルは仕様を記述する対象のもつ構造を表すために用いるとよい．たとえば，ソフトウェアのもつアーキテクチャの構造や，ソフトウェアが動作するメカ，電気的環境がどのような構成になっているのかを表す場合には，このモデルが向いている．

注3：それぞれのモデルの説明とあわせて紹介されている図は，その視点から描かれたモデルの一例である．同じ視点でも複数の表記法が存在するのが通常である．

▶データモデル

ソフトウェアの取り扱うデータに着目したモデル化(**図8**)である．このモデルは，仕様を記述する対象が複雑なデータ構造をもつ場合に用いるとよい．たとえば，データ処理を取り扱うようなシステムの場合(CCDから入ってくるデータをリアルタイムに処理し次のソフトウェアに渡すようなもの)，その取り扱うデータがどのようなものなのかを表すのに用いられる．

▶複合モデル

ここまでに紹介したさまざまな視点を複合化したモデル化である．このモデルは，仕様を記述する際に複数の視点をあわせて表現したい場合に用いるとよい．たとえば，**図9**左のモデルは，データモデルとアクティビティモデルをあわせたものである．

また，オブジェクト指向によるモデル化のように，システムの論理構造とデータ構造をあわせて表現したいという場合に用いるものもある(**図9**)．

▶述語モデル

ソフトウェアのふるまいを述語を用いて表したモデルである．これまで紹介したような絵として描かれたモデルの弱点である，条件やロジックの記述に用いられる．述語モデルは，記述された内容が矛盾していないかどうか検証可能であるものが多い(**図10**)．

▶数式モデル

ソフトウェアのふるまいを数式を用いて表したモデルである．センサから入ってきた値を元に，計算により出力値を決定するといったフィードバック処理を行うようなソフトウェアの仕様は，こうした数式モデルとして記述するとよい．

以上のようなモデル化を仕様の種類に応じて駆使しながら，仕様の記述は行われることになる．ここで，上記のモデルからわかるのは，一つの記述方法だけですべての仕様が記述できるわけではないということである．つまり，一つの記述方法(よくあるのはUML)にこだわるのではなく，ほかに仕様をきちんと記述できる手段があるのであれば，さまざまな記述方法を活用すべきである．

2.3 ツールの導入

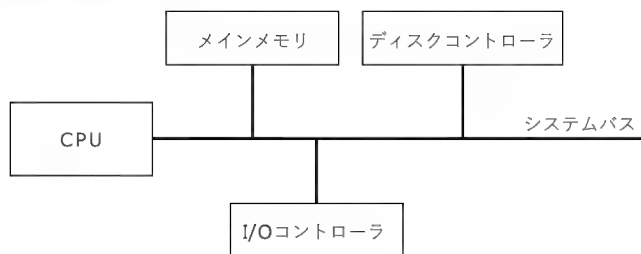
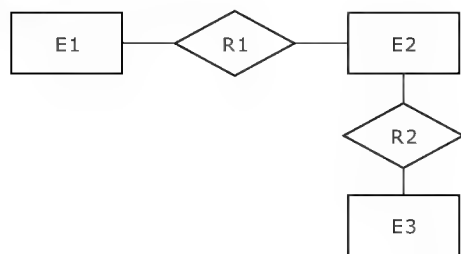
● モデリングツールの選択

ソフトウェア開発にはツールが不可欠である．組み込みソフトウェア開発では，実装以降のツールは豊富にあるが，分析，設計におけるツールはワープロソフトや表計算ソフトを使って記述しているところが非常に多い．グラフィカルな表記が中心となるUMLなどの導入を考えると，モデリングツールの導入を検討しているところも多いであろう．

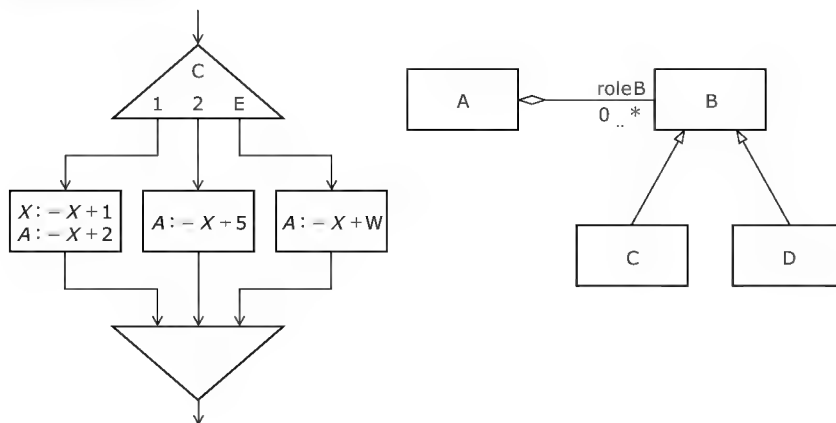
モデリングツールの選定に際して気をつけなければならないことがある．それは，「トレーサビリティ」をどのように考えるかということである．トレーサビリティとは，成果物間の情報の追跡可能性のことである．

つまり，分析→設計→実装と作られていく成果物のどことどこが対応しているのかがわかるということである．通常，プロセスを定義すると，このトレーサビリティを考慮してプロセスが作られることになる．そうしたトレーサビリティを実現するためには，成果物間に矛盾がないようにしなければならないが，そうした成果物間の矛盾をチェックするというのは，非常に手間のかかる作業となる．

モデリングツールに話を戻すと，モデリングツールには，「ドローツール系」と「CASEツール系」がある．ドローツール系(VISIO，DynamicDraw，SmartDrawなど)のものは比較的安価で，ステンシルなどを使えば，ほとんどのモデルを描くことができる．一方，CASEツール系(Rational Rose，MagicDraw，EnterpriseArchitectなど)のものは，モデルそのものが記述でき

〔図7〕構造モデル

〔図8〕データモデル

〔図9〕複合モデル

〔図10〕述語モデル

例：すべてのイベントはキューに追加される
Event(x)：xはイベントである
Queue(x)：xはキューに追加される
$\forall x\{Event(x) \Rightarrow Queue(x)\}$

るだけではなく，モデル間に関連付けをしたり，各モデルに対して付加情報(プロパティ)を定義することができる．

このドローツール系とCASEツール系のどちらのツールを選ぶのかは，トレーサビリティに重大な影響を与える．ドローツール系はたいてい，ただ絵を描くだけのツールなので，モデル間の矛盾のチェックといったことは行わない．しかし，CASEツール系の場合，描かれたものを絵としてではなく「モデル」として管理しているので，モデル間の矛盾を自動的にチェック・修正してくれるような機能がついていることがほとんどである．

つまり，ドローツール系を選択すると，安価な代わりにトレーサビリティの維持はユーザーまかせになる．一方，CASEツール系を選択すると高価な代わりに，トレーサビリティの維持を自動化できるというメリットがある．

最近では，モデリングツールというとコードの自動生成などに目をとらわれがちであるが，書くべき情報が記述できるかどうかということもモデリングツール選定の重要な要素である(**表2**)．

● ツールとプロセス

CASEツールやソフトウェア開発に用いられるツールの多くは，ツール開発者の強いポリシーの影響を受けていることが多い．CASEツールの場合，ツール開発者のポリシーにしたがわないことは，たとえUMLとして許されていることであっても表現できないことが多々ある．とくに，付加情報(プロパティ)のようにUMLで規定されていないものに対しては，非常に強く影響されている．

ツールを使用する際，ツールを自分たちのプロセスに合わせられればよいのだが，多くのツールではそうした柔軟性を持ち合わせていないことが多い．なので，ツールを選定する際には，ツールのもつポリシーが自分たちにとって受け入れられるものなのかどうかを十分に検討しなければならない．

そのためにも，ツール選定前に組織内のプロセスのどの部分にツールを適用するのか，ツールから出力された結果はプロセス内の他のアクティビティやツールとどのようにつながっていくのか(ツールチェイン)ということを考えなければならない．そうでないと，洋服に体を合わせるような事態(ツールに自分達の仕事の仕方を合わせること)を招くことになる．

2.4 構成管理

● 構成管理の不在

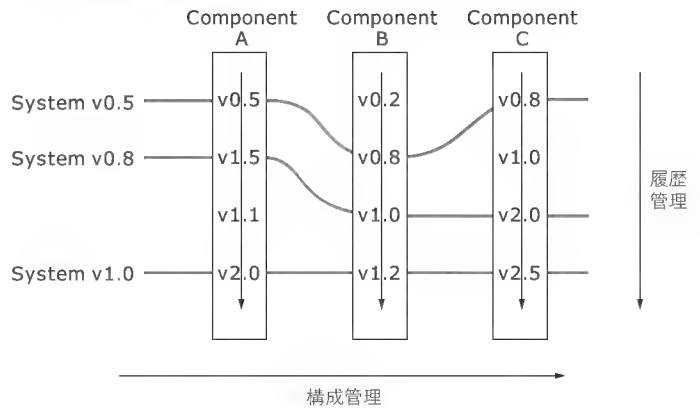
シリーズ開発が中心である組み込み系ソフトウェア開発では，類似システムが多く作られる．場合によっては，一つのソースコードからコンパイルスイッチによって複数の製品が作られることもある．そうした組み込み系ソフトウェア開発で重要になるのが構成管理である．

しかし，この構成管理が行われていない組織は驚くほど多い．構成管理を実施しているといっても履歴管理だけだったりすることもある．

ここで，構成管理について説明する．構成管理とは，ある時点のシステムがどのような要素で構成されていたかを管理する

〔**表2**〕**ドローツール系とCASEツール系の比較**

	ドローツール系	CASEツール系
トレーサビリティ	ユーザーがすべてを管理	ある程度ツールに任せられる
価格	安価	高価
付加価値	モデル以外の絵も描ける	コード自動生成機能やドキュメント自動生成機能

〔**図11**〕**構成管理の概念**

ものである．またそうした管理を実現する際に，各構成要素が時間によって変化するため，ある時点での構成要素の記録を残すために，履歴管理もあわせて行われることになる(**図11**)．

こうした構成管理は，ソフトウェアプロダクトライン的な開発を実現するためには必須の活動である．ソフトウェアプロダクトラインでのコア資産は，構成管理で管理されるソフトウェアの構成要素の一つであり，製品開発計画はそのまま構成管理計画となる．

● 構成管理の実施

ソフトウェアプロダクトラインのような開発を行うか行わないかは別として，構成管理は，組み込み系ソフトウェア開発において重要な要素である．とにかく，構成管理を始めるには構成管理環境を構築することが先決である．構成管理環境としては，オープンソースのもの(CVSなど)もあるし，専用のパッケージ(ClearCase，VisualSourceSafeなど)も販売されている．

構成管理環境を構築した後に決めなければならないのは，構成管理対象となる成果物である．よく構成管理というと，ソースコードのみを構成管理対象としているが，ある時点でのソフトウェアを構成しているのはソースコードだけでなく，そのソースコードとトレーサビリティをもつ成果物(設計資料や要求仕様)も構成物である．そのため，構成管理を実施する際には，どの資料を構成管理対象とするのかということを考えなければならない．

また，構成管理を実施するタイミングもプロセスとして決めなければならない．ここでいうタイミングとは，履歴管理としてバージョンやブランチを作成するタイミングと，構成管理として構成要素をまとめるタイミングである．そうしたタイミン

グが決まっていないと，構成管理は機能しなくなる．

3 技術管理

3.1 大規模複雑化するソフトウェア

数年前までは，組み込み系ソフトウェア開発は，一人ないし若干名で上流から下流，テストまでを担当するような小規模な開発が多かった．新たな開発は，前版に対する差分であり，同じ開発者でずっと同じシリーズを担当することも多かった．そのため，開発情報を広く他者と共有する必要はなく，そのため開発情報がドキュメント化されることなく現在に至っている組織も多い．

しかし，高付加価値な製品を開発するために，システムの多機能化や統合化が進み，近年では組み込み系ソフトウェアは急速に大規模・複雑化しており，大人数での開発を前提とするようなものも増えてきている．そういった状況では，担当者の入れ替わりも発生し，一人の開発者がずっと一つのソースコードをメンテナンスするということもなくなりつつある．また，そうした大規模開発では，情報の共有は非常に重要であり，システムの全体像や設計上のポリシー，一時的な制約に対する対処など，ソフトウェアを構築するために重要な情報が特定の開発者の頭の中だけに内在化されているという状況では，開発は困難なものとなってしまう．また，そうしたノウハウを内在化している特定の開発者が異動などでいなくなると，以降の開発がまったくできなくなってしまうという状況も起こりうるのである．

3.2 ノウハウの外在化

そうした状況にならないためにも，ノウハウを外在化するというのは組み込み系ソフトウェア開発を改善するために重要なポイントの一つである．

前述した，プロセスの定義もノウハウを外在化するための活動の一つであるが，ほかにもノウハウの外在化のための活動は考えられる．たとえば，ノウハウを収集するためだけのチームを作るというソリューションがある．そのチームはプロジェクト終了後に作成された成果物の分析や開発メンバーへのヒアリングを行い，ほかの製品開発でも利用できそうなノウハウを収集する．そうして集めたノウハウを整理し，誰もが利用可能なリポジトリに蓄積していくという活動を行うのである．

蓄積される対象となるノウハウとしては，分析，設計，実装，アーキテクチャのパターン，トレードオフの選択基準(技術的に二つの選択肢があった場合にどのような判断に基づいて選択を行うのかといったこと)，べからず集(アンチパターン)，メトリックス，といったものが挙げられる．

製品を作りっぱなしにするのではなく，こうしたノウハウを収集・蓄積し，それを活かすようなプロセスを定義することで，ソフトウェア開発を向上させていくことが可能となるのである．

いのうえ・たつき　(株)豆蔵
かわぐち・あきら　(株)ガイア・システム・ソリューション
さとう・けいた　(株)デンソー
はしもと・たかなり　ソニー(株)

第8章

組織・ビジネス面の取り組みポイント―― Agile方法論/ステージモデル/SECIモデル

開発効率化のための組織とAgile方法論

井上　樹/川口　晃/佐藤啓太/橋本隆成

組み込みソフトウェア開発の生産性，品質を向上させるために必要な要素の一つである，「組織・ビジネス面」について，技術者向けに解説する．いままで，組織面の話が，現場の開発担当者から言及されることは少なかったが，複雑化・多機能化してきた組み込みシステムを効率よく開発するために配慮すべき要素となってきた．本章では，開発効率化のための組織の分類から，最近話題のAgile方法論，チームやプロジェクトとしての発展のモデルであるステージモデルやSECIモデルなどを解説する．

（編集部）

第7章で述べた話は，製品開発自体を見直そうというものだった．この章では原点に戻り，なぜソフトウェア開発をするのかと問い直してみる．企業としてソフトウェア開発を行う理由は「ビジネスだから」である．ビジネスである以上，そこにはビジネス上の多くの戦略や障害および競争相手が存在する．

これらのことは，ソフトウェアサイエンスやエンジニアリングで解決する問題でないのは明らかだし，そもそもソフトウェアサイエンスやエンジニアリングは，ソフトウェア開発のビジネス上の課題の一つである「いかに効果的に開発するか」の問いに対する手段でしかない．それ以外の多くの課題には，まったく別の戦略や手段が要求される．企画や経営的視点からの企業の中長目標＆期戦略，人材育成，組織構成などである．

今後，ソフトウェア開発の生産性，品質の向上を求められるエンジニアおよび管理者なら，一段高い「企業人」としての視点が重要になってくる．なぜなら，企業として利益を上げるためには，かぎられた開発経費や資源（従業員や開発環境など含む）で企業全体の効果的な運営を行うには，企業戦略とリンクさせた多くのことを考慮して決定しなければならないからである．

はじめに――企業の競争優位には何が必要か

ソフトウェア開発を効果的に進めていくうえで，これまではCASEなどの開発環境のツールやオブジェクト指向技術，デザインパターンなどの技法，ソフトウェア開発のエンジニアリング的な側面に焦点を当てた開発方法論[注1]などが数多く登場し，開発現場に活用されてきている．

これらのツール，技法，方法論は，たしかにうまく導入すれば効果は上がるであろうし，実際に不可欠な重要技術である．そして，たしかに多くの成果をあげている事実がある．

しかしその一方で，当初期待したほどソフトウェアの開発効率や品質が向上していないことも事実である．最近ではCASEツールなどの開発環境やソフトウェア開発のエンジニアリングに焦点を当てた開発方法論だけでは効果は期待できず，ソフトウェア開発のプロジェクト管理面の重要性があらためて強調されている[注2]．さらにプロジェクト管理以前に，企業のビジネスビジョンや戦略が非常に重要であることも強調されている．

組み込み業界に目を移してみれば，携帯電話，自動車の車載システムなどに代表されるように，開発するシステムが複雑かつ大規模化するにしたがい，非常に多くの要素が関係してソフトウェア開発に影響をおよぼしている（**図1**，**表1**）．このような状況の中で，多くの努力が払われたのにもかかわらず，期待したほど成果が上がっていないという報告が多い．これまで常識と思われてきた方法が大きな問題を抱えていることも，明らかになってきている．

ソフトウェア開発に影響を与える要因の一つとして，エンジニアや管理者が在籍する開発プロジェクトや組織が挙げられる（**図2**）．企業や組織によってソフトウェア開発を行う組織構成やプロジェクトのことである．ここからは，まず組織やプロジェクトについて検討していくことにしよう．プロジェクトの価値や定義を明確にし，プロジェクトの特徴，プロジェクトが置かれている環境および部や課などの組織内の位置づけと関係について取り上げてみる．

「プロジェクトを成功に導くには？」と考えるとき，すぐにWBSやCPMといった具体的なマネージメント技法を連想しがちだが，まずは企業や組織の編成といった「構造」について検討してみる必要がある．プロジェクト管理は対象のプロジェクトの特徴，ビジネスドメイン，組織内，あるいは複数の組織にま

注1：代表的な開発方法論として，構造化手法ではワード・メラー法，オブジェクト指向ではOMT，Booch法，シュレイアー&メラー法などがある．以上の方法論はプロジェクト管理面も取り上げてられているものもあるが，大部分はエンジニアリングに中心を置いた方法論となっている．

注2：たとえばCMMは，米国カーネギーメロン大学のソフトウェア工学研究所（SEI：Software Engineering Institute）が開発したソフトウェアプロセス能力の成熟度のモデルである．

〔図1〕複雑化する組み込みシステム

〔表1〕ソフトウェア開発に影響を与えると考えられるおもな要素

企業・組織構造	市場の景気動向
人と人のコミュニケーション	協力関係にある企業との連携
企業のビジネス戦略	国際基準や法規による制約
競合企業との駆け引き	開発予算，資源などの企業の都合
従業員のスキルやモラル	etc.

〔表2〕機能型組織の特徴

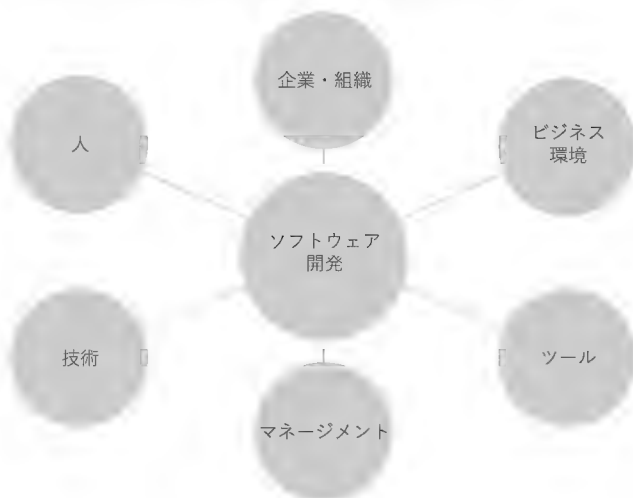
特徴	専門性や役割などの機能で従業員を構成する.「ライン組織」とも呼ばれる．安定した組織構成であり，各組織の専門性が成熟されていく．機能型の組織は日本企業に多く，比較的深い階層により構成される．機能型組織の特徴の一つとして，指揮系統が明確であることが挙げられる
長所	組織の再編成が少なく安定しており，組織ごとに専門性が明確になっている．専門性の成熟度を向上させることが容易．業務の多くが組織内で処理されることが多く，コミュニケーションが容易．組織の文化を築きやすい．組織の指揮系統もはっきりしている
短所	部門をまたぐ業務の調整では，柔軟性と対応のスピードの面で劣る．従業員のリソースを効果的に活用できない場合がある．新しいビジネスや環境の変化や要望に柔軟に素早く対応するには不向き．水平的なコミュニケーションが効果的に行われないなどの欠点も存在する

たがってプロジェクト構成がされている場合など，多くの複雑な要因のバランスの上に運営されており，近年はとくに多様化かつ複雑化している．これまでのソフトウェア開発で用いられてきたエンジニアリングは，プロジェクト管理やプロジェクトの特徴や取り巻くビジネス環境，企業や組織の文化をまったく，あるいはあまり考慮していなかったためか，ソフトウェア開発を取り巻く環境，前提や制約を考慮せずに，やみくもにCASEなどの開発ツールや方法論の導入およびプロジェクト管理を実施して効果をあげようと苦労している例が目立つ．

1 組織と組織マネージメント

1.1 「機能型」組織と「プロジェクト型」組織

企業で業務を進めていく枠組みには，大きく分類して「機能型」組織と「プロジェクト型」組織の組織構造がある．「機能型」組織の枠組みは，企業を「機能」による縦割型による組織の編成である．これを別のいい方で「静的な組織構造」という呼び方をする

〔図2〕何がソフトウェア開発に影響を与えるのか？

場合もある．機能型組織の例としては部署や課が代表的であり，経理部，総務部，人事部などが相当する．従業員を業務内容の専門性により振り分けて構成している．部門と呼ぶこともある．「機能型」組織の場合，通常は頻繁に組織の再編成を行うことが念頭にない場合が多く，定常的な組織編成になる．これは，ビジネス環境が頻繁に大きく変化しないことを前提としているためである．そこから「静的な組織構造」と呼ばれている．

もう一つの「プロジェクト型」組織とは，あるビジネス案件を達成するために，案件ごとに必要な専門性をもつ従業員から構成される組織構造である．これを別のいい方で「動的な組織構造」という呼び方をする．ある特定の業務を完遂するために，プロジェクトが編成され，目的が達成されれば，つまり業務が終了すればプロジェクトが解散されるからである．「機能型」組織が定常的な組織に対し，「プロジェクト型」組織は期間限定となる．

● **「機能型」組織の特徴**

「機能型」組織の特徴はいくつか存在する（**表2**）．従業員を業務内容の専門性で分け，それを事業部や課とする．つまり業務の種類や機能による組織の分割である．「静的な組織構造」と呼ばれるとおり「普遍性」という特徴もあり，一つ一つの業務に応じて組織を変化させることをせず，原則的には機能や業務内容で分類された組織構造内で業務に対応していくことになる（**図3**）．ただし，必要に応じて業務を複数の組織にまたがって遂行する場合もある．「機能型」組織の長所の一つとして，組織ごとに高度な専門性が蓄積されていき，組織および組織に帰属する従業員の専門性が成熟していくことが可能な点がある．

大きなもう一つの特徴として，「権限と命令系統の一貫性」が存在している．「権限」とは，責任をもち，下に指示を与えてその指示を実行させることを指している．「命令系統」は，組織が階層化させている場合に，組織に属する従業員が一人の上司から指示を受け（**図4**），反対に上司に報告を行う責任をもつことを指す．組織のトップからもっとも下層までの権限の系統が存在し，誰が誰に管理，報告するかが明確になっている．

〔図3〕典型的な企業の組織構造

〔図4〕典型的な企業の組織構造①：指揮系統

〔図5〕ライン組織のイメージ図

〔表3〕「プロジェクト型」組織の特徴

特徴	プロジェクトの業務を行ううえで必要な従業員が配置される．プロジェクトマネージャ（PM）による統率と管理下で制御される．プロジェクトリーダ（PL）は業務の具体的な牽引役である．プロジェクトのメンバは，企業の従業員としてどこかの部や課に属しているが，プロジェクトとしての指揮系統はPMになる
長所	新しいビジネスや環境の変化，要望などに柔軟に素早く対応するのに効果的である．プロジェクトの特徴，規模に応じて適切な従業員を活用しやすい
短所	プロジェクトのもつ一時的な性質により，従業員間のコミュニケーションや文化が育ちにくい．技術の成熟度が個人任せになりやすい．目的達成のためには合理的であるがプロジェクトメンバの考課や能力開発において長期的な育成をすることに課題がある．また，ライン組織の上司（ラインマネージャ）とプロジェクトマネージャ（PM）の二つの指揮系統が存在し，複雑になる傾向がある

最近は，急速に変化するビジネス環境に対応するために，「ライン組織型」（**図5**）の組織の位置づけや責任範囲が大きく変化してきており，プロジェクト管理を効果的に行うためには，従来の組織の枠組みを変化させる必要も出ている．

● 「プロジェクト型」組織の特徴

「プロジェクト型」組織は，機能型組織の対極にある組織構成である（**表3**）．まず成果を出すべき業務が存在し，その業務を達成させるために従業員が割り当てられる．業務が終了すれば，プロジェクトが終了し，解散される形態をとっている．

ただし，企業が強く「プロジェクト型」組織の構成を軸にして組織構成を行っていても，企業としての従業員の基本的な所属の場が必要である．このため，企業などのプロジェクトの上位に位置する組織は通常，部や課などの組織構造とプロジェクトという二つの異なる組織構造を組み合わせた形態となっている．プロジェクトと従来の会社組織の階層であるライン組織との関係は非常に重要である．プロジェクトとライン組織と関係に応じていくつものタイプに分類でき，タイプに応じてプロジェクト管理でのポイントが異なってくるからである．

1.2　組織～プロジェクト中心の組織編成

従来の「ライン組織」型から，「プロジェクト」型の形態が強くなればなるほど，従来の組織の枠組みが変化していく．また，プロジェクトの特徴やプロジェクト管理に応じて組織の位置づけが変化していき，組織とプロジェクトの関係は非常に大きな影響を互いに与えることになる．

近年は，ライン組織とプロジェクト組織の枠組みを融合した組織形態が多いといえる．現代的なプロジェクト管理では，「機能型」（**図6**）と「プロジェクト型」（**図7**）組織は「縦糸」と「横糸」のような関係になり，企業のビジネス戦略に応じて組織とプロジェクトの対応関係に応じて，大きくいくつかのパターンに分類されている．パターンに応じてプロジェクト管理上のポイントは異なり，プロジェクトを円滑に運営していくうえで重要となっている．

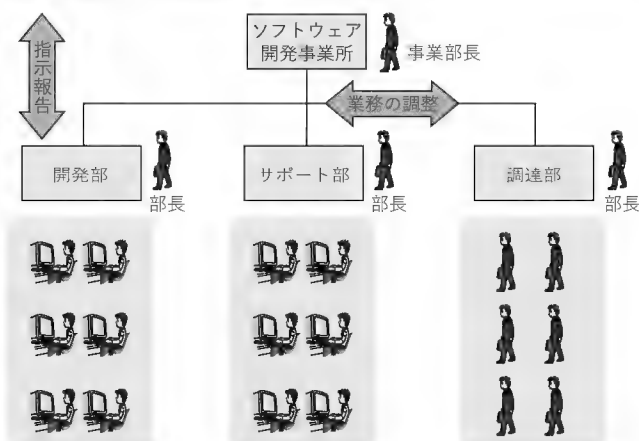
組織は指揮系統の基本枠であるので，組織階層が浅い，深いの違いが存在したとしても「統括」という，プロジェクトとは異なる管理機構と役職が存在する．一方，各プロジェクトにはPMというプロジェクト業務を行ううえでの指揮をとる責任者が存在する．「ライン組織」型と「プロジェクト」型のハイブリッド型では，利点と同時にいくつかの課題も存在している．

最近では，さらに「ネットワーク組織」という組織形態もあり，企業として取り組み始めているところもある．

● マトリクス組織

日本企業によく見られるピラミッド型組織では，各社員の能力が上手に活かせない，的確な評価ができないという問題点を抱えているといわれる．経営者が一部の幹部社員に対して指揮命令したことが下位の社員へと伝達されていくピラミッド型組織は，大量生産・大量消費の時代に向き，かつビジネスを取り巻く環境の変化が小さい場合に適していた．

〔図6〕機能中心型組織(ライン組織)の例

〔図7〕プロジェクト型組織

近年は，職能別組織とプロジェクト型を組み合わせたような組織形態である「マトリクス型組織」が，多くの企業で試みられている．ただし，指示・命令系統が複数になるため，組織としての意思決定が複雑になるなどの難しい課題も抱えており，必ずしも効果を挙げているばかりではないようだ(**表4**)．

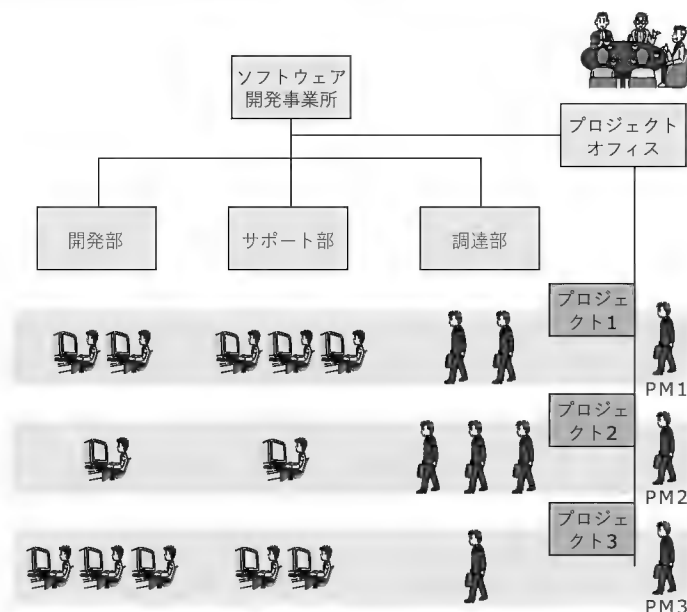
● **マトリクス組織のパターン1**
―― 弱いマトリクス型組織(部門調整型マトリクス組織)

ライン組織の権限が強いマトリクスの組織を「弱いマトリクス」(**図8**)という．アサインされたプロジェクトマネージャ(PM)は実質的に組織間のコーディネート役であるため，「部門調整型マトリクス組織」とも呼ぶこともある．この組織は，職能別の部門化が行われる職能部制組織と，製品別・地域別といった目的別の部門化が行われる事業部制組織の両方のデメリットを補完しつつ，それぞれのメリットを活かした組織形態となる[注3]．

また，各社員は実力と努力しだいでは年齢や役職に関係なく高い成果を上げられるようになるため，モチベーションの向上に効果的といわれる．優秀な社員についてはさらに重要な権限を与えることによって，効果的な人材登用が可能となる．

注3：製品，事業，地域，職能などを軸とした2ボスマネージャ制により，複数の命令系統が存在する組織．1960年代のアメリカの航空宇宙産業の企業において考案された．

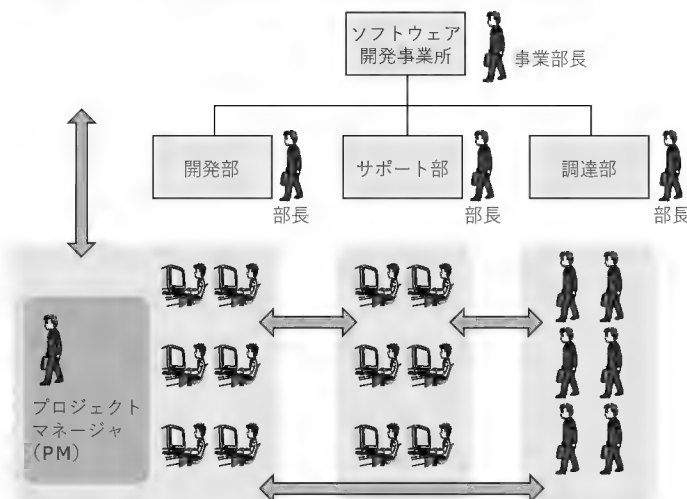
〔表4〕マトリクス型組織のメリットと問題点

メリット
① 効率性と柔軟性の同時達成
② リソースの効果的な配分と有効活用
③ 組織メンバのバランスのとれた能力開発
④ 組織内のオープンコミュニケーションの促進
⑤ 組織の協働関係の促進
問題点
① マトリクスマネージャ間の主導権争い
② 全社的レベルでマトリクスを維持するのが困難
③ 意見衝突が発生しやすい
④ 知識を蓄積するシステムが必要
⑤ 個人のストレスの増加

● **マトリクス組織のパターン2**
――中間マトリクス型組織(作業分担型マトリクス組織)

ライン機能とプロジェクト機能が同等で働く場合を「作業分担型マトリクス組織」(**図9**)と呼ぶ．PMはプロジェクト全体の管理を中心に活動し，顧客対応を行うことを主たる役割とするケースが多い．この組織は，プロジェクトに必要な要員確保とライン機能として必要な人材育成が可能だが，ライン長とPMの力関係が同等であるために，いったん組織上の問題が発生した場合には調整が困難になるというリスクもある．

● **マトリクス組織のパターン3**
――強いマトリクス型組織(リソースプール型マトリクス組織)

プロジェクト組織の権限が強く働くマトリクスの組織を「強いマトリクス」(**図10**)と呼び，プロジェクト側の要請に応じてラ

〔図8〕弱いマトリクス型組織――部門調整型マトリクス組織

イン組織よりプロジェクト活動期間中スタッフを貸し出す形態であり，PMはプロジェクト運営のためにメンバの役割のコントロールを行う形になる．プロジェクトが動的(ダイナミック)に構成される場合は，プロジェクト側の要請に応じるため要員のプールをする組織を別途に構えるケースも多くなっていく．

近年は，組織構造がネットワーク的なものへと変化してきている．マトリクス型組織が期待したほどの効果をあげていないことも理由の一つにある．組織は企業の経営戦略と深く結びついており，いかに効果的な組織構造を用いるかが大きなトピックとなっている．

2 「情報」から「知識」を活かした企業体質へ

ここからは，今注目されている**Agile開発方法論**にふれ，ソフトウェア開発の下支えをするいくつかのトピックを紹介する．Agile開発方法論では，従来のソフトウェア開発方法論やプロジェクト管理手法に欠けている，“人”にフォーカスをあてたものが多い．これは，ソフトウェア開発では，人のコミュニケーションや知識の共有が重要になるという主張からである．本章前半では組織やプロジェクトに関する解説を行ってきたが，どのタイプも人間の能力の成長具合やコミュニケーションなどが長所や短所になったことを思い出してほしい．なお，Agile開発方法論自身のくわしい解説は，参考文献5)などをあたっていただきたい．

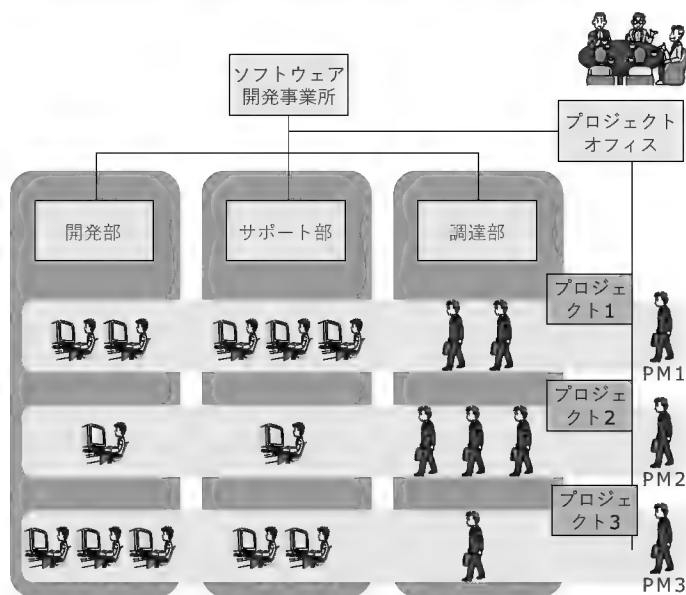
2.1 Agile方法論～科学的な理論や観察データとの比較

前述したように，最近Agile方法論が注目されている．これまでの官僚的な開発方法論やプロセスのアンチテーゼのようなイメージで，ソフトウェア開発者に歓迎されている．Agile方法論やプロセスは話題が先行しがちだが，非常に興味深い理論や各種検証からえられた情報を方法論の基礎としている．むしろ，これまで多く登場してきた官僚的な開発方法論やプロセスよりも非常に科学的といえる理論や観察に基づく検証を基盤にしているものも多い．

もし読者がAgile方法論を組織やプロジェクトに導入を検討しているならば，方法論やプロセスが基盤としている考えや哲学を正しく理解していると共感をもって導入や展開ができる．さらに効果も出しやすいだろうし，企業文化やプロジェクトの特徴に合わせ，最適な選択が可能となるのではないだろうか？

ここからは，Agile方法論が基盤としている理論や検証データなどの“一部”を解説していくが，これらの理論や検証データはもともと企業で競争優位を築く，あるいは保つために研究されてきたものである(それをAgile方法論が基盤としているところに方法論提唱者達のすごさが感じられる)．ぜひ参考にしていただきたい．

2.2 チームや組織の発展のモデル

多くのプロジェクト管理の本やセミナでは，組織やプロジェクトの種類と特徴を述べているが，導入を検討する前には多くのことを検討する必要があり，そう単純なものではない．現実には，プロジェクトや組織編成にはチームメンバ間の協調関係によるチームの成熟度と呼べる要因を検討することが必要である．なぜなら，「マトリクス組織」のところで説明したように，課題の多くは非常に人間的な感情や行動となっている．理屈的には「マトリクス組織」は理にかなっている部分も多いが，現実には人間は単純なものでないだけに，そうそう簡単にプロジェクトにすぐに順応できるとはかぎらない．

こういうことを学問的に研究する分野があり，「チーム(や組織)の成長のモデル」と呼ばれているものが存在している．モデ

〔図9〕中間マトリクス型組織 —— 作業分担型マトリクス組織

〔図10〕強いマトリクス型組織 —— リソースプール型マトリクス組織

ルとして主なものに，古典的なモデルのTuckmanの“ステージ・モデル”やGersickの“Punctuated Equilibrium(中断された平衡説)モデル”などがある[8]．

“ステージ・モデル”とは，チームは段階的なステージを経て発展・成長していくというモデルである(**図11**)．ステージは，(1) Forming，(2) Storming，(3) Norming，(4) Performingとしてモデル化されている．1番最初は「Forming」でチームが構成される．そして2番目の「Storming」チーム内にいろいろな混乱や問題が発生するというのである．混乱や問題は，多くは人間関係や文化や哲学に関する議論である．3番目の「Norming」で議論の後，チームとして一体感が出てくるという．最後の4番目の「Performing」は，いよいよ活動に成果が出始め，チームとして結果を出すというものである．ここでは，チームという言葉が使われているが，企業の組織やプロジェクトと置き換えて考えても問題ない．

このTuckmanの“ステージ・モデル”を聞いたとき多くの方は，「なるほど」と思えるだろう．しかし，Gersickはこのモデルに異を唱えてPunctuated Equiliblium モデルを提案している．多くの多様なチームを観察した結果，チームのライフサイクルの約半分あたりで大きな変化が訪れるのだと主張している．

チームは初期にチームとしての体制ができあがり，しばらくはあまり大きな変化を見せないという．しかし，ライフサイクルの半分あたりから，ここまでの共有経験から活動の進め方，どうあるべきかが活発に議論され，チームのターニングポイントが訪れ，それ以降もチームが解散するまで継続するという．このPunctuated Equiliblium モデルでは，時間という概念が重要な役割を果たすことがポイントである．

参考文献8)では，外部コンサルタントを利用する際にチームの最初の段階か，ライフサイクルの中間あたりが最適であり，逆にそれ以外の時期は避けるべきであり，チームにもこのようにある程度『慣性の法則』が存在すると説明している．このことから，Punctuated Equiliblium モデルは，プロジェクトチームのマネージャやプロジェクトマネジメントなどにも重要な示唆を与え，チームのライフサイクルをどのように設定したらよいのか，そのライフサイクルの中で，いつ，どんな行動をしたらよいのか，などを考える際の一つのフレームワークになると付け加えている．

2.3 企業の競争優位と組織の知識創造

● 知識創造へのパラダイムシフト～リエンジニアリングの限界

1980年代後半から米国企業は「日本的経営」を分析し，リエンジニアリングという横型マネージメントを行う考え方を導入したが，成功したとはいえない．原因はいろいろある．その中に現実の仕事のやり方，プロセスというのは，社員一人一人のメンタリティとか，組織のもっている文化，風土そういうものが深く関わっているというのがやはり大きい．

これが意味することは，一昔(二昔以上？)にはやったリストラとリエンジニアリングは，フラットな組織構造を目標に，徹底的に組織のスリム化を始めたが，リストラのターゲットは中間管理職が多く，その結果，機能しなくなった企業および組織内の業務がうまく回らなくなったという企業が多数発生した．結果からいえるのは，とくに中間管理層を削減すると，組織の中で情報，知識が流れなくなるのではないかということである．中間管理層は表面上は，何をやっているのか明確でない業務も多いが，どうやら組織やチームの中で，知識や情報を共有するうえで重要だというということが見えてきたのだ．

データベース，コンピュータ情報などはストック/検索ができるが，じつはなかなかコンピュータに載らない情報や知識があり，これは多くの場合，人間に体化されているということがしだいにわかってきた．この点をふまえていないと，今はやりの「ナレッジマネージメント」も形だけの魂のないインフラになってしまう．「仏作って魂入れず」といったところだろうか？　なお，このまさに重要な知識，情報が流れなくなったことを“コーポレート・アルツハイマー”と表現するらしい．

さて，ここからXPや多くのAgile方法論が土台としている理論による今後のソフトウェア開発に重要な話を進めていく．XPやAgile方法論についてご存知の読者は，XPやAgile方法論の「原則」や「プラクティス」を考えながら読んでみてほしい．必ず「あぁ～などほど．そういうことか」と感じるのではないだろうか．

〔図11〕ステージ・モデル

● **知識創造の論理**

知識には二つのタイプの知識がある．一つは「形式知」という言葉になる，あるいはドキュメントになる知識である．もう一つは「暗黙知」というなかなか言葉や言語，ドキュメントに表現することが困難な知識である．この違いは何かというと，「形式知」は分析的に生み出せる知，ディジタル的なものであり，共有可能な知であるのに対し，「暗黙知」は経験の集積の中で獲得する知，アナログ的なものであり，完全な個人知，一人一人異なるものということである．また，西洋では論理分析的な「形式知」を重視するのに対し，東洋，とりわけ日本では個人の経験，感覚に基づく「暗黙知」を非常に尊重する．

知識創造とは「形式知」，「暗黙知」(**表5**)のどちらが正しいという議論ではなく，両方の知が同等に必要で，両者は相互補完の関係にあり，「暗黙知」，「形式知」のスパイラル運動がたいせつであるということ，つまり「暗黙知」が「形式知」に，さらには「形式知」が「暗黙知」に変換され，それがスパイラル状に上昇したときに，知は組織的に生み出されるという考え方である．

● **知識創造の論理～知識社会の競争へ**

ソフトウェア開発がなぜハードウェアの生産のようにいかないかということが議論されて久しい．近年は，ソフトウェア開発とハードウェアの生産を同じように考えること自体が間違いとする意見が優勢になりつつある．これは，これから説明する「SECI」モデルを通じて説明ができるのではないだろうか？　まずは，これから始める解説を読んでいただきたい．最後に，XPやAgile方法論のプラクティスと比較して，整理を試みたい．

● **知識創造の二つのタイプ――「暗黙知」，「形式知」**

「暗黙知」というのは完全に個人の知である．「暗黙知」は，「思い」と「ノウハウ」ともとらえられる．「暗黙知」は，SECIモデル提唱者の野中教授によれば，日本がもっともこの「暗黙知」をたいせつにしている国民だと指摘し，経験で道をきわめるというのは，おそらく世界で日本だけではないかとも付け加えている．

人間は経験をしないと，あまりたいしたことは理解できない．しかし，経験を何らかの手段を使って形式化しないと進歩もない．いつもでも同じことを繰り返すことになるからだ．経験から得た結果を分析し反省し，フィードバックもできない．つまり「暗黙知」は永遠に個人知なので，議論をし，それを「形式知」に変換して特殊なノウハウを普遍化しないかぎりは組織やチーム(プロジェクト)の知にならない．

ここから重要になってくるのだが，**『「知の創造」とは「暗黙知」と「形式知」の「スパイラル運動」』**である．「暗黙知」が「形式知」に変換され，「形式知」がさらに「暗黙知」に変換され，それがスパイラル状に上昇したときに，知は組織的に生み出されるという考え方だ．組織の知が豊かになると，その組織に所属している社員一人一人が次々に新しい経験に挑戦して新しい「暗黙知」を獲得する，あるいは「形式知」を次々に組み合わせて新しい「形式知」を次々に生み出すという循環が生まれる．これが，野中教授が提唱する「SECI」モデルで，国際的に非常に有名だが，日本のソフトウェア業務に携わっている方々には案外知られていない．この野中教授が提唱する「SECI」モデルが，外国の研究者やエンジニアに支持され，ソフトウェア開発方法論やプロセスに組み込まれて日本にいわば「逆輸入」されるのは複雑な思いである．

〔表5〕暗黙知と形式知[5)]

種　類	説　明
「形式知」(Explicit Knowledge)	文章，言葉，数学的な記述などを使って，明確に表現できる知識．第三者への伝達や第三者による利用・評価が容易にできる
「暗黙知」(Tacit Knowledge)	ある個人や組織の内部には存在するが，明示的に表現できず，第三者が利用したり，評価することが容易にできない知識．「暗黙知」には，熟練工が持つ技能や芸術的な感性などが含まれる

〔図12〕SECIモデル[9)]

● **知識創造プロセス～「SECI」モデル**

「SECIモデル」(**図12**)の第1は『「暗黙知」から「暗黙知」をつくる』で，これを共同化(Socialization)と呼んでいる．つまり，ここでは言葉を使わないで「知」をつくるということ，基本は共体験である．典型的なものとして日本の伝統的な親方と徒弟の関係がある．内弟子制度が残っている相撲や落語では，親方と徒弟の関係というのは弟子が親方の家に住み込んで，共体験，生活のリズムを感知しながら観察，模倣し，そして自分でやってみる．企業でいえばOJTや先輩が手取り足取り指導するチューター制もこれの範疇に含まれると考えられる．

第2は『知のつくり方は「暗黙知」から「形式知」をつくる』だが，これを表出化(Externalization)という．ノウハウを言葉にする，あるいはコンセプトにすることにあたる．

第3は『「形式知」から「形式知」をつくる』である．これを連結化(Combination)と呼び，言葉やコンセプトを組み合わせることをさす．

第4は，『「形式知」から「暗黙知」をつくる』になる．これを内面化(Internalization)と呼び，言葉やコンセプトを体得するということ，つまり，分析的に頭でわかったことを自分のものにする，経験の反復，道をきわめるということ，かみしめながら

行動することにあたる．たいせつなのは，ここで説明した1～4をスパイラル状に回すということである．これがSECIモデル（**図12**）である．

▶共同化

共同化を促進するには「暗黙知」から「暗黙知」をつくるので，自分も「暗黙知」を獲得するためには実際に手を使って作業することが，そしてその『場』で『知』を体で体感することが重要になる．これが身体で知識，情報を獲得するプロセスとなる．

▶表出化

個人のノウハウをうまく形にし，伝えることが重要だ．ノウハウというのは多くの場合，アイデアとかイメージであり，イメージをうまく表現する方法が必要なわけである．そのため，メタファとかアナロジというものが非常に重要になる．うまい比喩やわかりやすい話が，「暗黙知」を「形式知」に変換するときには有効となる．

▶連結化

第3の連結化には「形式知」の伝達，普及が必要だ．当然のことながら，ここではコンピュータがたいへん効力を発揮する．

▶内面化

行動，実践を通じた「形式知」の体化が必要で，例としてはOJTなどがある．

おわりに

ハードウェアの生産は，ボルトやナットから複雑な装置まで図面で正確に表現できる（知識の形式化）．しかしソフトウェアは見えないだけに，形式化しにくい．また，Agile方法論の中には，これまでの多くの方法論がソフトウェア開発をハードウェア開発と同様のプロセスであらねばならないとする仮定が，誤りであると断言しているものも存在する．参考文献5）では，CMMの生みの親でもあるHumpphreyがCrosbyの書いた*Quarity of Free*の中で紹介されているMMM（Manufacturing Majority Model：製造成熟度モデル）をソフトウェア開発に取り入れた事実をあげ，この誤った仮定のために以降の20年間は，まさにこの「呪縛との戦い」だったといっている．以上のようにソフトウェア開発は，開発に参加する開発関係者の「暗黙知」の影響をとても強く受ける作業の連続だからだと解釈できる．この点からXPやAgile方法論の多くは，短い反復開発や開発関係者との直接のコミュニケーション，毎日のミーティングや実際に動作させて評価するなどのことを重視している．

また，これまでは，形に見えるものばかりの合理化や形に見えることをめざしてきたが，それがそもそも困難であり，合理的なアプローチと呼べないことがSECIモデルなどから理解できる．今後，ソフトウェア開発の発展や企業の競争優位を築くために，従来とは異なる新たなアプローチを検討していく必要がありそうだ．

参考文献

1）ジャンカルロ・スッチ他著，小野剛他訳，『XPエクストリーム・プログラミング検証編』，ピアソン・エデュケーション・ジャパン
2）橋本隆成，「Agileなソフトウェア開発」，『Software People』,Vol.1，技術評論社，2002
3）橋本隆成，「実践！CMM導入徹底ガイド」，『Software People』,Vol.2，技術評論社，2003
4）山田正樹，「知識創造とソフトウェア開発」，『Software People』,Vol.2，技術評論社，2003
5）Schwaber Ken，Mike Beedle，*Agile Software Development With Scrum*（Series in Agile Software Development）
6）原田 保，楠木 建，関西生産性本部 著，『超社員術－会社に依存しない自律創造型「仕事人」への道』，英二出版
7）野中郁次郎，竹内弘高 著，梅本 勝博翻訳，『知識創造企業』，東洋経済新聞社
8）チームはどのように発展していくのか：Gersickのグループ発展モデル“Punctuated Equilibrium Model”，http://www.geocities.co.jp/WallStreet/4716/teamdevelop.htm
9）キーノートスピーチ：「組織的知識から社会的知識創造へ」，野中郁次郎，http://www.glocom.ac.jp/itplat/archive/0405sympo/panel1/2_nonaka_1.html
10）「日経デジタルエンジニアリング」4月号，http://dm.nikkeibp.co.jp/free/nde/kiji/no204/report08.html
11）組織的知識創造論の新たなる地平――最新の理論展開をめぐって――，http://www.shc-creo.co.jp/webcreo/bn/e19980201.html
12）野中郁次郎ほか，『企業の自己革新～カオスと創造のマネージメント』，中央公論社
13）高木晴夫，大久保隆弘，『企業維新――ネットワークリーダーシップが組織を変える』，ダイヤモンド社
14）Stephen P. Robbins，高木晴夫ほか翻訳，『組織行動のマネージメント――入門から実戦へ』，ダイヤモンド社

いのうえ・たつき （株）豆蔵
かわぐち・あきら （株）ガイア・システム・ソリューション
さとう・けいた （株）デンソー
はしもと・たかなり ソニー（株）

第9章

組み込み系ソフト開発効率向上のための人的な面での取り組みポイント

人的協働モデルとモデル作成者の適性

井上　樹/川口　晃/佐藤啓太/橋本隆成

ソフトウェア開発の効率や品質を左右する大きな要素の一つとして，個々の技術者の適性があげられる．本章では，3種類の人材を有効に活用して上流の分析・設計モデルを効率よく開発するための人的協働モデルを紹介し，その考え方のさまざまな要素を解説する．

（編集部）

ソフトウェア開発の主体は「人」であり「組織」である．本章では，プロジェクト横断的に「人」や「組織」の戦略を立てる**SEPG**（**Software Engineering Process Group**）やシニアマネージャを対象として，現在筆者らが研究中の，モデル時代の開発組織戦略を紹介する．

製品全体の品質が向上し，また市場が飽和状態にある今，多機能化や統合・複合化により製品に高付加価値を付けた競争力ある製品の開発に生き残りをかけている．これにより製品やソフトウェアの開発自体が大規模化し，また開発の軸足がハードウェアからソフトウェアにシフトしてきたように，ソフトウェアからサービスやコンテンツにシフトしてきている．

これをソフトウェア開発に当てはめると，下流の実装を重視していた開発から上流の分析やアーキテクチャ設計を重視した上流へのシフト（**図1**），つまりコードからモデルへのシフトとなる．これは過去，組み込みソフトウェア開発においてアセンブラ言語からC言語に移行したようなパラダイムシフトだ．

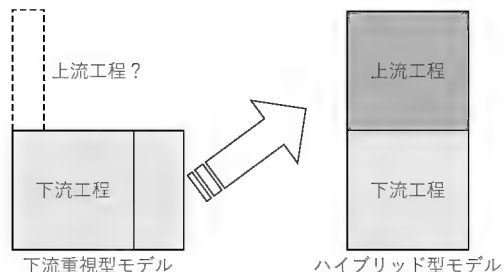
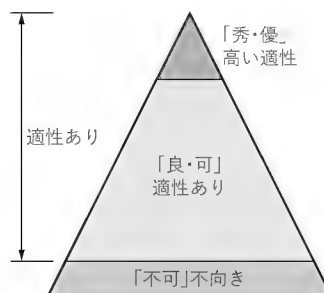
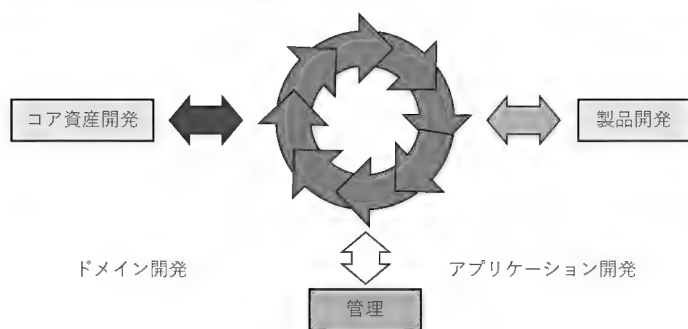
今日のC言語のライブラリやガイドラインなどの資産は一朝一夕に完成できたわけではなく，試行錯誤による長い間の積み重ねでできたものである．そして今，これをモデルに対して行おうというわけである．

現在は，誰にでもある程度の品質でC言語による開発ができている．ただし，これはライブラリやガイドラインといった環境が整備されているからであり，このような環境が整備されていなかったC言語の初期では，人（技術者）による質のばらつきが大きかった．環境は，C言語に適性のある技術者によって作り続けられてきた．モデリングにおいても，これと同じことがいえるであろう．つまり，初期では人によるばらつきが大きく，これを一定の質にするためには，適性のあるものが環境を作り上げることが求められる．

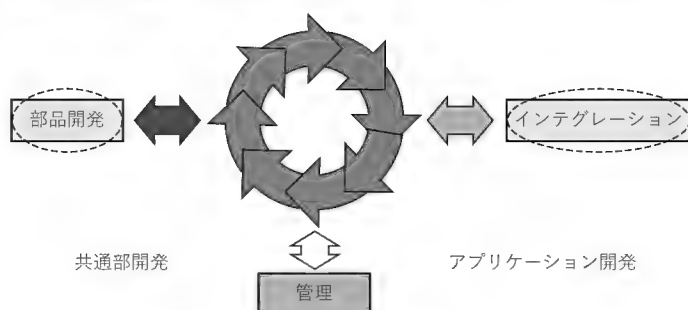
ソフトウェア開発は，人による創造的な活動である．したがって，向き不向きがある．筆者の経験上，ソフトウェア開発のそれぞれの工程において不向きだといえる人は少数であり，それ以外の多くは，なんとか動かせるレベルにまで作り上げることができる．そして同時に「良い」ものを作れる高い適性をもった人も少数である（**図2**）．

適性の有無はプログラミングやモデリング以外にも，マネージメントやテストなどすべての活動にいえることである．本章ではモデル開発者を対象とするが，モデル開発に適性がないといっても技術のすべてを否定するつもりはなく，きっとそういう人材は，プログラミングや教育など他の活動に高い適性があるはずである．不向きな活動をさせるよりも，その人材の適性に合わせて活動をさせる，まさに適材適所で人を配置することが品質や生産性の向上，そしてさらにモチベーションの向上につながるのである．

本章では，現在筆者らが研究中である，モデル作成の適性に応じた3種類の人材を有効に活用して，上流の分析・設計モデ

〔図1〕人的モデル構造

〔図2〕適性マップ

〔図3〕プロダクトライン開発プロセス

〔図6〕部品開発とインテグレーション

〔図4〕共通と個別（1）

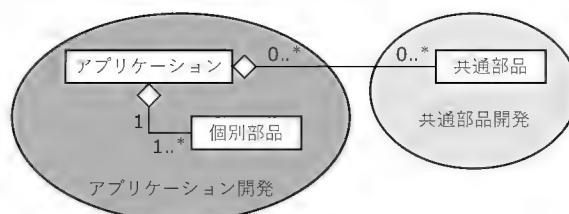
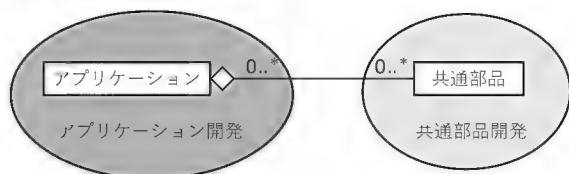
〔図5〕共通と個別（2）

ルを効率良く，しかも高品質に開発するための人的協働モデルを紹介する．また，モデル作成者の適性についても言及する．

1 人的協働モデル

適性をもつ人材を，すべてのソフトウェア開発に配置することは困難である．しかし幸いにも，組み込みソフトウェア開発はプロダクトライン[1), 2), 3)]，つまりシリーズやバリエーションといった差分開発の形態をとることが多く，異なる部分よりも共通する部分（プロダクトラインではコア資産と呼ばれる）が多いために，開発の適性をもつ人材をコア資産開発に集中させることが可能となる．

プロダクトライン（**図3**）では，共通部品であるコア資産は，開発の適性をもつドメイン開発者によって開発される．しかも開発される共通部品は，通常の開発よりも時間とコストをかけてテストが行われるために非常に質の高いものとなる．

アプリケーション開発者は，アプリケーションごとにこの共通部品と（自ら開発した）アプリケーション固有の個別部品を統合してアプリケーションの開発を行う．実際に，このような取り組みをしている組織も多い．

プロダクトラインでは，コア資産，つまり共通部品の利用に関して利用促進や利用状況の監視，そして部品開発者へのフィードバックといった「管理」という活動が設けられているが，このような方式で実際に開発をしている組織では，この「管理」という部分が行われていない場合が多く，これが次のような問題を招いている．

アプリケーション開発者が既存の共通部品を理解し，その中から適当な部品を探して，他の部品と苦労して組み合わせるといった労力は，実際の作業だけではなく精神的な負担も大きい．利用する共通部品の割合が多いほど全体の質は向上するために，アプリケーション開発者はできるかぎり新規に個別部品を作らないように努力すべきであるにもかかわらず，アプリケーション開発者は他人が作ったわけのわからない部品を苦労して使うよりも自ら新規に個別部品を開発することを望む．

これにより共通部品として使われるものは徐々に少なくなり，同じような個別部品が組織として管理されることもなく，結局は再利用前の状態に戻ってしまうことになる．

プロダクトラインでは「管理」により，アプリケーション開発者による部品開発を抑制する．しかし，アプリケーション固有部分の開発は依然として適性のない開発者によって行われることになり，全体としての質にばらつきが残ってしまう．

「共通」と「個別」という考え方を少し変えてみよう．共通と個別の違いは，共通が二つ以上（実際には，あらかじめ利用を想定して作られた部品もあるので0個以上）のアプリケーションで利用されるのに対し，個別はたった一つのアプリケーションにだけ利用されるということである（**図4**）．

個別として開発した部品が他のアプリケーションから利用されれば共通部品となり，また共通部品を利用していたアプリケーションが一つだけになったとき，共通部品は個別部品とみなされるのである．

ならば一つと二つ以上を区分けせずに，すべてを共通部品と考えてもよいのではないだろうか（**図5**）？

こう考えると，個別部品を含めたすべての部品は共通部品（コア資産）とみなされて共通部品開発者によって開発されることになる．アプリケーション開発者は必要な部品の開発を（開発の適性がある）共通部品開発者に発注し，開発された部品を組み合わせて評価するといった，コア資産のインテグレーションに仕事がシフトされる（**図6**）．

このように開発の適性により作業を分担することによって，ア

プリケーション開発者は不向きなソフトウェア開発そのものの作業から解放される．しかも，作業が単純化されるために国内外へのアウトソーシングや，ツールによる自動化をも検討できる．この枠組みによって作業が軽減されてできた「ゆとり」を高付加価値なサービスの追求やビジネスアーキテクチャの検討に当てることができる．しかし，アプリケーション開発の負担は軽くなる一方で，共通部品開発者の負担はますます重くなる．開発量だけではなく共通部品開発では最新技術の評価・導入や将来の変更を先取りし，通常よりも品質や性能といった難易度の高い活動が要求される．いくら能力のある技術者とはいえ，一人でシステムの全体から詳細までを担当することはできない．

ここで，開発を「方向付け」，「推敲」，「作成」，「移行」の四つのフェーズに分けて反復させるUnified Process（**図7**）を参考に，共通部品の開発を分業させる．

分業は「良いモデル」を作成できる高い適性をもった者と，「動くモデル」を作成できる通常の適性をもった者で分ける．高い適性をもった者には共通部品全体に関わる基本的な戦略を「方向付け」と「推敲」で決定させ，一般的な適性をもった者には高い適性をもった者が定めた戦略にしたがって「作成」と「移行」で実際にアプリケーションとして利用できるレベルにまで作り込ませる．

以降，便宜上，高い適性をもち，共通部品，アプリケーションやシステム全体の戦略を考える者を「基本開発者」，「基本開発者」の戦略にしたがって，アプリケーションやシステムで使う部品を開発する者を「共通開発者」，さらにアプリケーションやシステムの要求仕様に合わせて（共通部品開発者によって開発された）共通部品を組み合わせ，テストをする者を「個別開発者」と呼ぶことにする．

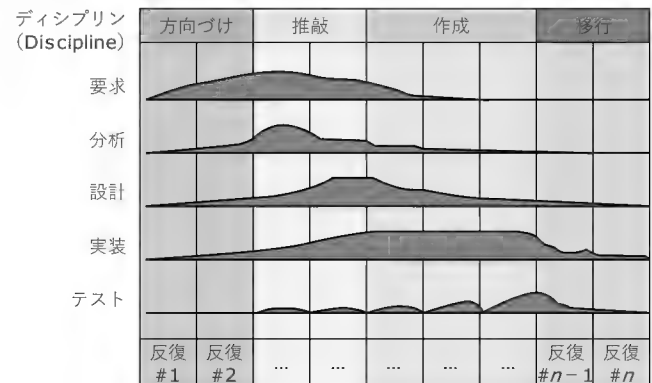
2 基本開発

方向付けと推敲フェーズでは，若干名の適性がとくに高い開発の目利き，スペシャリストによってアプリケーションの共通部品開発向けの技術戦略を基本開発モデルとして開発する．ここで一般的なUnified Processと異なるのは，対象が特定のシステムやアプリケーションではなく，それらが含まれるロードマップ，つまりプロダクトラインだということである．

2.1 方向付け

ドメインエンジニアリングにおける方向付けは，開発組織に横断的なソフトウェア開発方式の改善を任務とするSEPG（Software Engineering Process Group）の仕事と重複する（基本開発者とSEPGは非常に近い役割だが，現段階で，この二つの違いを明確にすることは難しい）．「基本開発者」は，技術スペシャリストという立場でSEPGと協働して以下を実施する．

- SEPGが定義するリスクに加えて，技術者の視点でシステムのアーキテクチャの候補，およびシステムで新規部分や難しい部分などを洗い出す
- リスクが解消できるかどうかを検討する

〔**図7**〕**Unified Process Discipline**

- 構築しようとしているシステムが，SEPGなどによって定められたビジネスゴールを達成し，現状の課題を克服できるかを実証する
- 市場動向や技術動向を先読みし，変化点（ホットスポット）と，安定点（フローズンスポット）を予見する

以上を実施し，また検証することを目的としてプロトタイプを作成する．ただし，このプロトタイプは上記の目的を達成するためだけに作成されるもので，作り捨てるものである．開発組織からオブジェクト指向やUMLの導入に際し，コンサルティングの要請を受けるが，ほとんどはここでのプロトタイプ開発に対するものである．

しかし，先にあげたようなリスクや，課題，そしてゴールが対象となる開発組織で明確になっていない場合，試行に対する評価基準が曖昧となり，その嬉しさが見えずに終わってしまうことになる．これはコンサルタントにも問題があり，コンサルタントはいわれるがまま指導をするのではなく，ゴールや評価基準の定義に積極的に関与することが必要であろう．

2.2 推敲

推敲フェーズでは，以下の活動により，対象ドメインに属するすべてのシステム，アプリケーションに共通する技術戦略を共通開発者が高い精度で計画，実現できるようになるまで高める．

- システムのアーキテクチャと関与者や外部システムにとって重要な機能を網羅するアーキテクチュラルベースラインを作成する
- 基本となるアーキテクチャやパターンを構築する．このとき，過去の開発資産を理解し，有効に活用していかなければならない
- 方向付けフェーズで洗い出されたホットスポットとフローズンスポットに対する変更を容易とする機構をアーキテクチャにあらかじめ仕込んでおく
- 重大なリスクを洗い出し，軽減させる
- 信頼性や応答時間などの性能といった非機能要件を設定する
- 個別開発向けに共通開発が用意するソフトウェアモデル，プロセス，ガイドライン，教育，そしてツールなどに対する要件や要望を作成する

ソフトウェアモデルの多くは，基本的なパターンの派生で実現される．基本開発は，この基本パターンを定義する活動であり，共通開発は基本パターンにしたがって派生を作る活動となる．基本開発の遂行フェーズで，それぞれの基本パターンに対するサンプルを作成することによって共通開発での基本パターンの理解と派生作成を促進させることができる．

3 共通開発

共通開発では，基本開発で定義された基本モデルを個別開発で利用できるレベルまで発展させていかなければならない．共通開発においてとくに考慮すべき事柄は基本開発で定義されているため，共通開発では基本開発の指示にしたがって共通開発モデルを作成していけばよい．

3.1 作成

共通開発の作成は，従来のシステムソフトウェア開発に相当する．ただし，ドメインエンジニアリングであるために，すべてのアプリケーション個別の要求を，この段階で知ることは困難であり，また，すべてのアプリケーションで共通して利用されるため，通常よりも高い性能，品質を実現しなければならないことが特徴である．

● **インクリメンタルな要求**

プロセスや，手順，ツール，教育などは，そのドメインに対する汎用的なものとして構築し，提供することができるが，アプリケーションごとに異なる要求仕様すべてに対応できる共通なソフトウェアモデルをあらかじめ開発することは不可能である．そこで，まずは過去に作成されたアプリケーションを再構築し，同等の機能をそろえる．そして，将来に対する分は，基本開発モデルとして提供される市場動向や技術動向をあらかじめ仕込んでおく．これ以外に個別開発から五月雨式（さみだれしき）に依頼されるようなアプリケーション個別の要求には，基本開発で柔軟に対応できるようなアーキテクチャが提供されるので，それにしたがって対応すればよい．

● **高い性能と品質要求**

リアルタイム性や品質を実現するためには，高度な理論および技術が必要とされる．これをすべての開発者に求めるのは現実的ではない．これが必要とされるのは，ずばり共通開発者である．リアルタイムに関する技術的な話題は，すでに多くの情報があるので割愛する．

3.2 移行

個別開発者用の共通開発モデルができたからといって，共通開発者の仕事は終わりではない．共通開発者は，個別開発者が共通開発モデルを利用する際の支援までを活動範囲とする．つまり導入の教育や，導入後の技術サポート，また，場合によってはアプリケーション開発のレビューなどに参加し，共通開発モデルが適切なレベルにあるかを常に把握しておかなければならない．ただし，これはプロダクトラインのように（共通開発モデル）管理専門の担当を設けてもかまわない．ただし，その際には，（共通開発モデル）管理担当者は共通開発者と密接に連携を取らなければならない．

4 個別開発

個別開発では，基本構成について予定されていた変更箇所に対し，アプリケーションの要求に合った部品（オプション）を組み替えることでアプリケーションを構築する．これはインターネットを使ったパソコン販売や，最近流行のコーヒーショップ，そしてメーカーオプションやディーラオプションなどがある自動車販売でもおなじみの形式であり，CTO（Configure To Order），「受注仕様生産」，「注文仕様生産」と呼ばれる．

このようにソフトウェア以外の世界では，CTOやCCP（Channel Configuration Program）が実用化されてきている．そして，ソフトウェア開発においても下流を海外にアウトソーシングしたり，ツールにより自動化することが検討されており，CTOやCCPに向かっているようである．

実際の個別開発は，以下のような活動となる．

a）個別開発者は，過去のアプリケーションモデルから対象アプリケーションにもっとも近いモデルを探す
b）探したアプリケーションモデルとの違いを調べる
c）異なる箇所に対し，最適な部品を探し，組み替える．もし，

Column
CTOとBTO/CCP

CTOと関連するものとしてBTOとCCPがある．

BTO（Build To Order）は「受注生産」と呼ばれ，顧客の注文を受けてから製品を生産することを指す．顧客は，カタログに用意されたラインナップの中から希望製品を選択してオーダーする．メーカーは，オーダーに基づいて生産する．パッケージソフトウェア以外のほとんどのソフトウェアがBTO方式に該当する．このBTO方式では，顧客の細かな要望に応えることはできない．

CTOは，標準製品ではなく基本製品を用意しておき，ユーザーが提示する仕様にしたがって基本製品に周辺装置を追加したり，ソフトをインストールして製品を組み立てる方式のことを指すものであり，洋服のイージーオーダーに似ている．CTOは製品仕様にユーザーの要求を反映させた製品を届けるという点がBTOと異なる．その点でCTOは，ユーザーの顧客満足度向上にもつながるしくみだといえる．

CCP（Channel Configuration Program）は「チャネルアセンブリ」と呼ばれ，ユーザーの希望する仕様に合わせて製品を組み立てる方式でという点ではCTOと同じだが，組み立てをCTOの場合はメーカーが行うのに対し，CCPはディーラなどの販売チャネルが行う点が異なる．

〔図8〕**人的協働モデル**(基本形)

〔図9〕**人的協働モデル**(応用形)

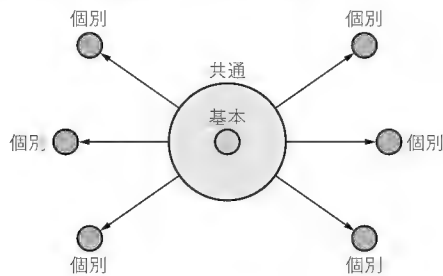
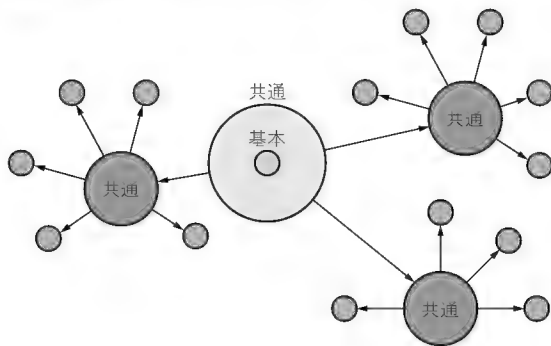
〔表1〕**開発体制**(基本形)

	基本	共通	個別			
			製品A	製品B	製品C	製品D
システム分析						
アーキテクチャ設計						
ソフトウェア分析						
ソフトウェア設計						
ソフトウェア実装						
テスト						

〔表2〕**開発体制**(応用形)

	基本	共通	ドメインα			ドメインβ		
			共通	個別		共通	個別	
				製品A1	製品A2		製品B1	製品B2
システム分析								
…								

〔表3〕**対象範囲**

	基本	共通	個別			
			製品A	製品B	製品C	製品D
システム分析						
アーキテクチャ設計						
ソフトウェア分析						
ソフトウェア設計						
ソフトウェア実装						
テスト						

最適な部品がない場合には共通開発者に部品を発注する

d) 部品をインテグレーションする

e) 結合，統合テストを行う

と，実際の活動は，検索，比較，組み換えが中心となる．これは部品構成の変更であり，構成管理ツールを使って実現できる．

4.1 応用モデル

ここであげた人的協働モデルは，一つの基本・共通開発グループと，複数の個別開発グループで形成される(**図8**，**表1**)．

個別開発の個数が増えてくると，基本開発と共通開発の負担が増してくる．だからといって，現在多くの組織で行われているように開発グループ単位に基本開発と共通開発の組織を形成すると，開発グループ間での共有が抑制されることになる．開発組織は製品ごとや顧客ごとに形成される場合が多いが，ソフトウェアモデル的な分類とは異なるはずである．そこでCCPの方式(**図9**，**表2**)を取り入れ，事業的視点の組織とソフトウェア的視点の組織の融合を図る．

5 モデル作成者の適性

開発体制の枠組みができたら，次に考えるのは中に入れる人材である．ここでは，筆者らが研究中の以下の事柄に関して記述する．

- それぞれの活動で行うこと
- その活動に対する適性および評価基

ソフトウェア開発には，モデル開発やプログラミング以外にも多くの工程があり，それぞれに担当者がいて，また，適性の基準もそれぞれ異なる．筆者らの研究はモデル作成者の適性だけではなく，ソフトウェア開発に携わるさまざまな役割に関しての適性を研究していくことをテーマの一つとするが，まずはもっとも緊急性を要するモデルを作成する(システム，ソフトウェア)分析者とアーキテクトから研究に取り組み始めた(**表3**)．管理者に対する適性やトレーニングも重要だとよくいわれているが，これには開発以外のさまざまな視点が必要とされるために，次回以降の取り組みとする．

5.1 モデル作成者の活動と適性

ここでは，組み込みソフトウェア開発においてもっとも緊急性を要する分析およびアーキテクチャ設計を行うモデラに焦点を当てる．

少し前まで組み込みソフトウェア開発は，一人や二人で行うような小規模な開発が多く，分析や設計などの工程をスキップし，いきなりコードで考えることも少なくはなかった．

しかし最近では，携帯電話の開発に見られるように競争力の高い製品作りのために高機能化や統合化が増えてきたため，システムが大規模化/複雑化し，一部の変更がほかへの思わぬ影響を巻き起こすことになってきている．

これに対し，大規模・複雑なシステムを見通しが良く変更に柔軟に対応できるシステムに再構築する要望が高まっている．これは膨大かつ複雑なシステムを整理し，将来の拡張に備え拡張・変更が容易なアーキテクチャに整頓する能力をもつ技術者によ

って成し遂げられる．

このような能力はトレーニングによる後天的なものよりも，むしろ先天的なものである可能性が高いが，今のところ筆者らはこれに対する根拠をもっておらず，今後の重要な検討課題となっている．もし後天的なものであればトレーニング方法の確立によって解決されるが，先天的なものであれば，人材を適材適所に配置するために適性を明らかにしなければならない．トレーニング方法の確立は重要なテーマではある．しかし，これについては技術教育の専門家たちが取り組んでいるので，筆者らは先天的な人材に対する適性を選び，検討をしている．

適性を考える前に，モデル作成活動に対する基本，共通，そして個別の活動は**表4**のようになる．

ここで重要なのは，基本，共通，個別それぞれに入力となるモデルや資産があり，入力モデルの理解およびその適用能力が要求され，多くの方法論や技術書などが前提としているようなまったくの新規にモデルを作成するということは稀だという点である．

次に，それぞれの活動に対するスキルを考えてみる．

個別開発者

表4からは，個別開発者は「類似検索」と「相違特定」，そして「構成変更」がおもな活動となり，これらはツールによって支援が可能である．したがって，共通開発において適切な環境が構築できれば，個別開発には特別なスキルは不要と考えられる．

共通開発者

共通開発者は，個別開発のように完成された道具（ツール，プロセス，手順書など）や教育がなくとも，ある程度の教育や道具があれば共通開発を行うことが可能である．この共通開発がカバーする範囲は従来のソフトウェア開発にほぼ等しく，従来のソフトウェア開発者であれば，共通開発を行うことが可能だと考えられる．ただし，共通開発モデルはすべての製品に組み込まれることから，影響範囲が非常に大きく，いままで以上に確実にリアルタイム性や品質に関する技術などを習得することが前提となる．

基本開発者

共通開発者は，不十分ではあるが教育や道具を必要とした．基本開発者は，自分たちが使う道具や教育は整備されていなくとも共通開発者向けの道具や教育そのものを開発していかなければならない．このように，何もないのに等しい状況より，情報や共通性を見つけ出し，また，先を予見して対処する，いわばパイオニア的な能力が要求される．

〔表4〕モデル作成者の活動

役　割	活　動	入　力	出　力
基本開発	●既存開発資産とドメインを分析する ▶既存開発資産の共通性と相違性，ポリシー，リスク ▶市場動向，技術動向，業界動向 ●共通開発者向けにパターンを創出する	既存資産	基本モデル
共通開発	●基本開発モデルを基盤として既存開発資産から再構築する ●相違部分を部品化する ●個別開発者向け環境を構築し，洗練する ▶ガイドライン，ツール，教育など ●個別部品を汎用化する	既存資産 基本モデル	共通モデル
個別開発	●対象システムにもっとも近いシステムを検索する（前版または基本版） ●検索したシステムとの相違点を調べる ●相違している部分に対応する部品を検索する ▶該当部品がない場合には共通開発者に発注する ●検索して得られた部品を入れ替える ▶部品構成の変更→構成管理	共通モデル（コア資産）	システム

5.2 基本モデル開発者の適性

基本モデル開発者には，どのような人材が向いているのであろうか？　**表4**で示した活動からは，類似と相違を見分ける分析能力，情報が残されていない既存資産の開発時のポリシーを想像する能力，また汎用化するための抽象化能力などが求められそうである．筆者らは，このような仮定のもと，さまざまな年代，さまざまな技術経歴をもつ技術者や学生に対してモデリングをしてもらい，データを分析している．

● **筆記によるモデリング**

最初に，どのようなモデルを良いと感じるかを調べるために，（文章で書かれた）自動販売機やエレベータなど，組み込みの複数の標準問題を四角と線でモデリングしてもらった．先天的な素質を見きわめたかったので，被験者はオブジェクト指向やUMLの経験がほとんどない人たちである．モデリングに先立って，以下のようなヒントを与えた．

- あるシステムに対する複数のモデリングを示した．これを示すことによって，モデリングは，視点ごとに異なることを気付かせるねらいがあった
- 抽象的なものもいくつか混ぜておき，その価値に気付いて自分のモデルに活かせるかも興味の一つだった

このときのモデル作成は，モデルを作成するときのこだわりや視点を知ることを目的としたものであり，モデルの正しさは求めていない．もっとも多かったモデルは，**図10**のように問題文にある物理的なものをCPUから中央集権的に線を結んだものであった．

〔図10〕典型的なモデル

それ以外に，**図11**，**図12**など，いくつかのモデルが筆者らの目にとまった．コンサルティングや教育で正解を求める人は非常に多いが，これらのモデルに見られるようにモデルは唯一ではなく，視点の違いによって異なる．しかも，視点は対象となるシステムのタイプや非機能要件，プロジェクト事情，ビジネス戦略，そして顧客との関係により，さまざまとなる．良いモデルを作成できる人は，無意識のうちに仮想的に視点を設定しているものと思われる．

これらのモデルを並べ，筆者らはどこに魅力を感じたのかを話し合い，まず「抽象力」が挙げられた．抽象力は次の三つに分類できた．

- 問題文にある言葉だけでモデリングされている
- 問題文にない「物理的」な抽象モデル（外部通信モジュールなど）が含まれている
- 問題文にない「論理的」な抽象モデル（販売処理，発注処理など）が含まれている

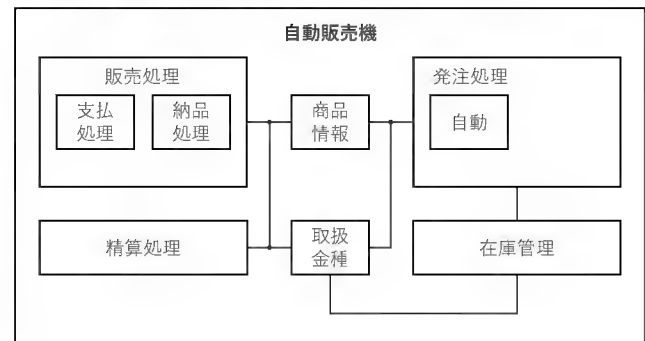
そして，良いモデルに共通していえたのは，どのモデルも見やすいということである．関係の近いものを近くに置き，影響範囲によりパッケージ化している．これは頭の中でモデルが整理整頓できていることを意味する．しかも，左から右へ，上から下へと空間を非常にうまく使ってモデルを表現している．このほかにも，モデリング能力の基準となるポイントは残されているが，それは今後の課題として検討を継続する．

● **演習形式によるモデリング**

モデリングは最終形式だけではなく，考える過程も重要である．これを調べるには演習形式でモデリングを行い，課題を変更したり，ヒントを出したり，他者のモデリングを見せる．そしてそれから何を感じ，変更にどう対処するか，人のモデルの良さを理解し，自分のモデルをどう洗練させるか，そのモデリングの過程を見ていけばよさそうである．

● **トレーニングへの適用**

このように何がよいモデルか，どのようにモデルを作成するのがよいかを解き明かせれば，高い適性をもつ人によって行われるモデル作成を非属人化できる教育が考えられるであろう．

〔図11〕売買に着目したモデル

おわりに

ソフトウェア開発は人間による知的創造活動である．適性を見きわめ適材適所に配置するだけで無駄を削減でき，生産性や品質が向上する．しかも，これにより開発者のモチベーションが向上し，開発者のやる気によって計算以上の効果をもたらすことになるであろう．

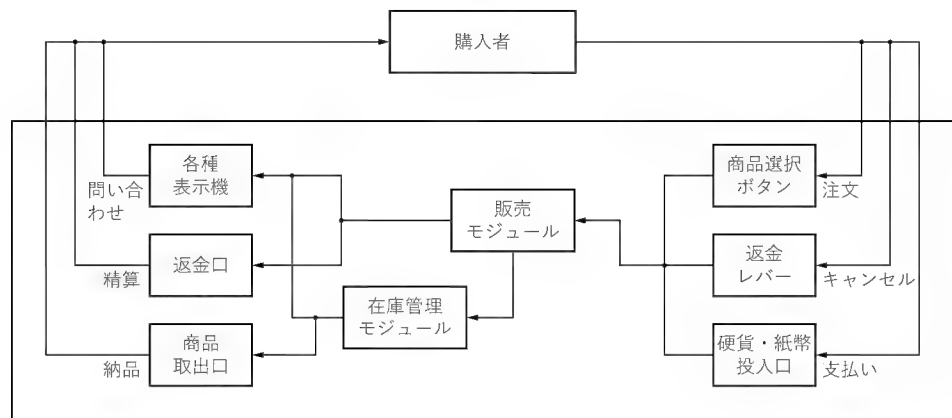
また，属人的活動は，高い能力をもつノウハウを体系化，外在化し，環境を作り上げることによって初めて非属人的活動となる．このノウハウの体系化，外在化，および非属人的環境の整備に筆者らは取り組んでいる．

参考文献

1）http://www.sei.cmu.edu/plp/
2）Paul Clements, Linda Northrop, *Software Product Lines: Practice and Patterns*, Addison-Wesley, 2001
3）佐藤啓太，今関剛，「品質と生産性向上のためのプロダクトライン入門」，『Software People』，Vol.1，技術評論社，2002
4）イヴァー・ヤコブソン，グラディ・ブーチ，ジェームズ・ランボー著，日本ラショナル（株）訳，藤井 拓監修，『UMLによる統一ソフトウェア開発プロセス』，翔泳社，2000

いのうえ・たつき　（株）豆蔵
かわぐち・あきら　（株）ガイア・システム・ソリューション
さとう・けいた　（株）デンソー
はしもと・たかなり　ソニー（株）

〔図12〕外部アクタとのインターフェースに着目したモデル

Epilogue

組み込み機器開発効率化のための今後の取り組み

井上　樹/川口　晃/佐藤啓太/杉浦英樹/橋本隆成

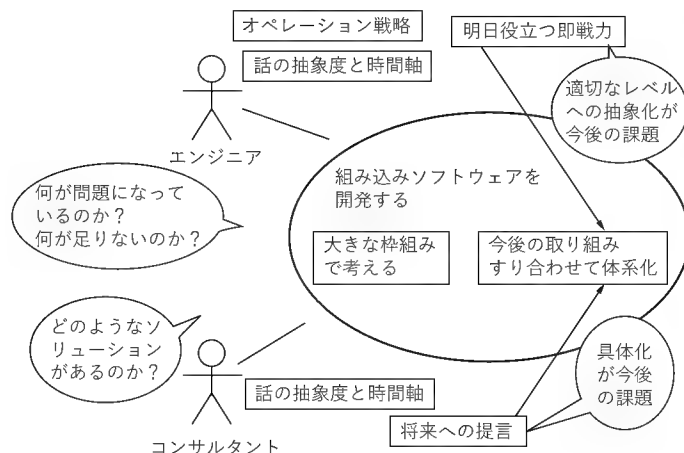
本特集では，「今，組み込みソフトウェア開発で何が問題になっていて，それに対してどのようなソリューションがあるのか」を中心に，前半部分は開発の一線で活躍しているエンジニアから現場での活動について解説した．後半部分では，組み込みソフトウェアのコンサルタントから，現場からは一歩下がった視点での取り組みポイントを解説した．本章では，この二つの視点での意見をすり合わせ，問題意識が一致している点，ずれている点，今後の取り組みなどをまとめる．**図1**に，概略を示す．

1　開発現場とコンサルタントの問題意識が一致している点

1.1　大きな枠組みで考えるということ

今回，もっとも心配していたのは，開発現場，コンサルタントとも，今，直面している目先の問題に目を奪われ，大きな枠組みで議論できないのでは？ということだった．実際には，開発現場側の報告は，事例紹介，組み込みにおけるオブジェクト指向の活用などの技術的な話題に加え，投資効果の測定やパラダイムシフトの説得材料など，ビジネス/オペレーション的な話題もカバーされていた．また，視点も一企業の枠組みにとらわれない大局的な観点からまとめられ，多くのエンジニアにとってリファレンスとしての意味をもつものだったと考える．

また，コンサルタント側の取り組みポイントの提案も，技術，組織・ビジネス，人の視点から，今後の組み込みソフトウェアの取り組みポイントを大枠でおさえており，事業としての組み込みソフトウェアの将来を見据えた提言になっていると考える．したがって，大きな枠組みで事業としての組み込みソフトウェアをとらえるという視点は共有できていたと思う．

〔図1〕今回の活動概要

2　開発現場とコンサルタントで問題意識がずれている点

2.1　話の抽象度と時間軸にずれあり

ある程度，事前に予想していたことであり，予定どおりでもあるが，やはり話の抽象度がずれていた．開発現場からの報告は，非常に実践的な話，即戦力の報告が中心になった．時間軸的にも，明日の開発をどうすれば少しでも改善できるのかという内容である．開発現場に身を置く読者の皆さんには非常に参考になる内容だったと想像する．

一方，コンサルタント側の話は，一段抽象度が高く，また時間軸も数年後を視野に入れた提言に近い内容だった（技術的な取り組みポイントに関しては，即戦力な話題もあった）．コンサルタントは，実際の事業から一歩退いた立場にあることもあり，現場の開発者に比べると（ある面）冷静に課題を分析することになる．今回のコンサルタント側の記事は，この視点でまとめられたもので，改善の方向性をよく示していたと思う．両者のとらえ方は，どちらが良いというものではない．両方の特性を活かしつつ，いかに実践的なソリューションを作り上げるかが今後の課題といえる．

2.2　具体的な提案への落とし込み（コンサルタントの課題）

ただし，開発現場からは，「コンサルタントの皆さんには，もっと現場の状況を直接見てほしいと思っています」とのコメントをいただいている．コンサルタントの次の課題は，本特集で示した改善の方向性を，開発現場が有効活用できる具体的な抽象度に落とし込み，開発現場に提案することにある．また，開発現場の課題を適切な抽象度で整理することも必要になるかもしれない．

2.3　組織内でのオペレーション戦略

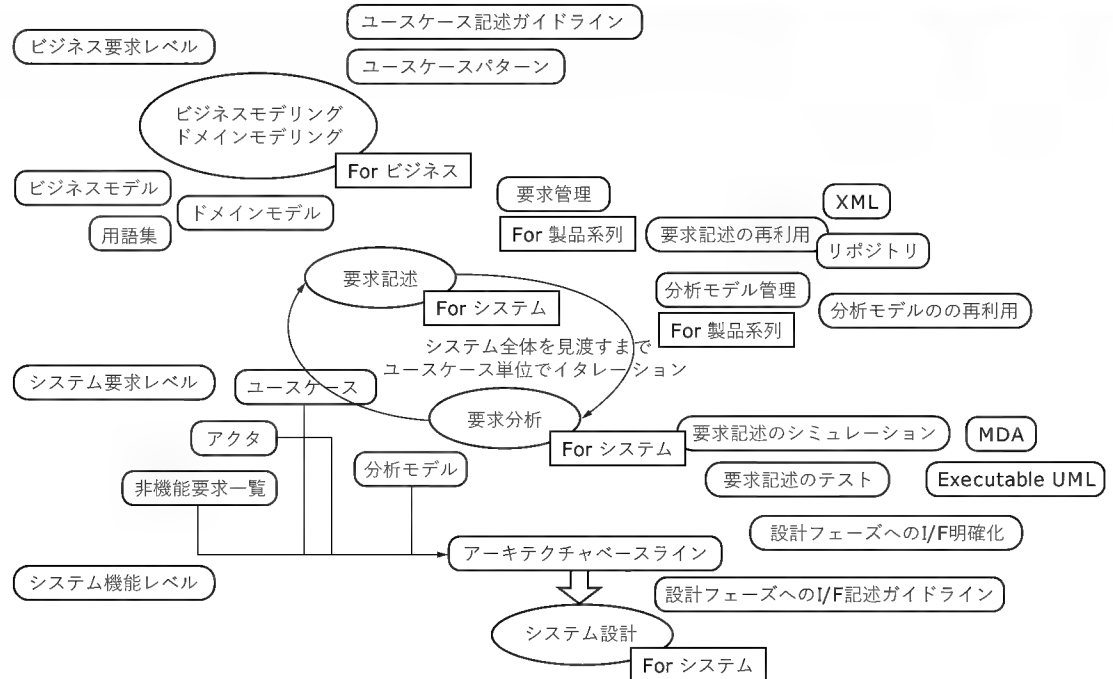
開発現場からの報告には，企業や組織内で関係者をどのように説得するのか，その説得材料としてどのようなデータが必要かなど，組織内でのオペレーション戦略もしっかり言及されていた．コンサルタント側は，数年後のゴールは明確に描けていたが，現実問題として，どのようにしてそのゴールにたどり着くのか，今回は誌面の都合もあり，その戦略が十分に示されていなかったように思う．今後の活動で，ゴールに向けての戦略を明確化する必要がある．

3　今後の取り組み

今後は，開発現場から報告された開発の現状と，コンサルタント側が提言した取り組みポイントを統合する活動を進めたい．開発の現状に対して，コンサルタントが提言した取り組みポイントを具体化し，ソリューションにまとめあげる作業になる．

3.1　課題のモデル作り

たとえば，**図2**は，今回の特集執筆にあたり，試みとしてまとめた

〔図2〕組み込みシステム開発のモデル

組み込みシステム開発のモデルである．このモデルは技術に着目した内容になっている．このようなモデルを，組織・ビジネス面，人の面，管理面で作成する．また，組み込みソフトウェア開発のどのフェーズ（たとえば，要求，設計，実装など）の課題なのか，対象ドメインはどこなのか（たとえば，家電，事務機，自動車など）も加味したモデルにする必要がある．

3.2　ソリューションを検討，マッピング

次に，各課題に対してその解決方法をマッピングする．これに，解決できる時期（すでに解決済み，1年後，数年後，解決のメドなしなど）を合わせて**図3**のようなマップを作成する．このマップのことを，筆者らは「ソリューションマップ」と呼んでいる．

3.3　ソリューションマップの意義

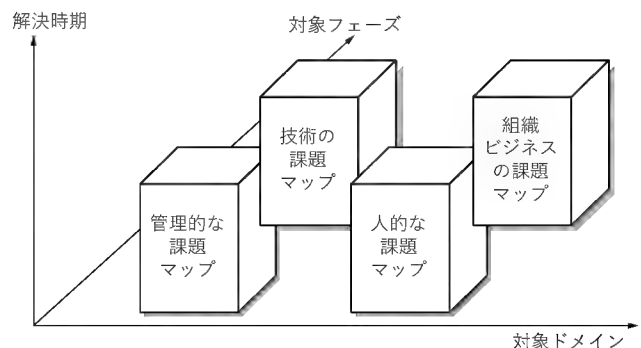
上記の要領で作成されたソリューションマップには，次のような意義があると考えている．

- **企業の開発者にとっての意義**
  標準的な技術レベルの確認／自分の取り組みの正しさを確認／将来の方向性を確認／開発に役立つ情報の入手／取り組むべきテーマの掘り起こし／キーパーソンとしてのブランド価値向上／企業と大学の協働を促進
- **コンサルタントにとっての意義**
  ビジネスとしてのソリューションの体系化：重要な技術をもれなく体系化し，コンサルティング活動に役立てる
- **アカデミックな意義**
  共通ボキャブラリの整備／保有研究成果のマッピングと広報／将来の研究テーマの掘り起こし／企業と大学の協働を促進する

3.4　ソリューションマップの課題

マップにしたことによって，新たにどのような価値が生じるのか，マップで見られる利点を明確にする必要がある．

3.5　取り組みの枠組み

上記の活動に関しては，情報処理学会ソフトウェア工学研究会の組み込みワーキンググループやSESSAME（組み込みソフトウェア管理者・技術者育成研究会，http://blues.tqm.t.u-tokyo.ac.jp/esw/）に働きかけ，産学協同で推進することも検討している．

〔図3〕ソリューションマップの作成

おわりに

一般的にいえば，組み込みソフトウェアは製品を構成する一つの部品である．事業の構造としては，組み込みシステムという製品を売って収益を得るしくみで，組み込みソフトウェア単体の付加価値は明確でない．

しかしながら，一つの独立した事業として組み込みソフトウェア開発を考えるなら，まず，その付加価値を明確にしなければならない．付加価値が明確になってはじめて，その最大化への道筋が見えてくる．

筆者らの活動が，組み込みソフトウェアの付加価値を計るための手助けになれば幸いである．

いのうえ・たつき　(株)豆蔵
かわぐち・あきら　(株)ガイア・システム・ソリューション
さとう・けいた　(株)デンソー
すぎうら・ひでき　富士ゼロックス(株)
はしもと・たかなり　ソニー(株)

組み込みプログラミングノウハウ入門

第11回 スポラディックスケジューリングのはなし

藤倉俊幸

はじめに

五月雨(さみだれ)的仕様変更という言葉がある．五月雨とは，今の五月ではなく旧暦の五月に降る雨で，梅雨のことである．梅雨というといろいろなイメージがあるが，五月雨的とか五月雨式とかいう場合は，途切れがちに繰り返すことをいう．つまり，途切れがちな仕様変更がいつまでも繰り返されるということを五月雨的仕様変更と呼んでいる．

五月雨攻撃というと，たとえば戦闘機が編隊を組んで攻撃をすることではなく，一機または数機で攻撃を繰り返すことである．五月雨戦術は，労使交渉などで使われる，繰り返しながら交渉を引き伸ばす戦術である．いずれも量は少なめで，ときどき途切れるが長く続くことをいっている．

蝉時雨(せみしぐれ)の時雨とは，秋の終わりごろの降ったり止んだりする雨，あるいはそのような曇りがちの暗い空模様のことである．「しぐれ」の「くれ」は暗しからきていて，「し」のほうは暫(しば)しで，ちょっとの間暗くなるときに降る雨のことを指すようだ[注1]．

五月雨は停滞前線の低い高度の層雲から降るシトシト雨で，時雨は対流層を移動する積雲からザァッと降る雨らしい．それで蝉時雨はどのような蝉の鳴き方かというと，ひとしきり鳴いて止む鳴き方を指す場合と，真夏の林でアブラゼミの鳴き声が雨のように降り注ぐことを指す場合，どちらかというと夏も終わりの頃の夕暮れのヒグラシの鳴き方を指す場合があるようだ．絶え間なく鳴き続けることを指す場合もあるが，これは時雨の意味を逸脱している気もする．時雨は日本海側に特有なので太平洋側の人には実感が湧かないため，誤用されているのかもしれない．

もう一つ村雨(むらさめ)という雨がある．これは，突然降り出すかなり強い雨のことを指す．やはり繰り返す．南総里美八犬伝に出て来る名刀「村雨」は，チャンバラの最中に火事を消すこともできる．つまり火事も消えるほどの激しい雨が，突然降り出すのが村雨である．繰り返しを含んだ代表的な雨の降り方3通りであるが，日本には雨に関する言葉が多いので，このほかにもあると思う．繰り返すといっても周期的というわけでもなく，また完全にランダムでもない繰り返し方を表現する例としてあげてみた．

米国のtelecommuterと呼ばれる，会社に行かずにインターネットを使って仕事をする一種の在宅勤務者は，2002年で約2,500万人いるそうである．彼らは，喫茶店などで電話回線を使って仕事をしている．そういえば，筆者も最近会社に行かずにPHSを使ってノートPCからインターネットに入って仕事をすることが増えてきていて，なしくずし的テレコミュータになりつつある．そのときの通信トラフィックを見ると，五月雨的に通信しているときと時雨的に通信するとき，あるいは村雨的にオーバロード気味に通信しているときがある．

時雨状態のときにコンパイルなどを始めると通信が切れてしまう．村雨状態のときでは，PCが黙り込んでしまうので最悪の場合には再起動しなければならない．したがって通信中は通信状態を見ながらPCを使うようにしている．ネットワークに接続された組み込み機器もこのような負荷にさらされている．ただし，リアルタイムOS(RTOS)を使っている場合は，時雨や村雨状態になっても雨が終わればまた動き出す．いままではこれだけでRTOSは堅牢でたいしたものだと思っていたが，マルチメディア機器などに応用が広がるにつれ，どんな時でも機能することが要求されるようになってきている．

さてそうなると，周期的ではないがときどき繰り返す，まとめて起こるイベントをどうさばくかが組み込みシステムでも重要になる．とくに最近の組み込みシステムのように，単純な機器制御だけでなくネットワークにつながって通信をするような場合には，イベントを周期イベント(periodic event)と非周期イベント(aperiodic event)に分類するだけでは対応しきれない．雨の降り方にもいろいろあるように，もう少し多様な分類を考える必要がある．そこで今回は，非周期イベントのモデル化のアイデアと，それに対応するスケジューリング法を紹介する．

1 イベントモデル

リアルタイムスケジューリング関係の本でイベントモデルというと，**図1**の四角で囲った部分に示したイベント到着モデルが説明してある．周期的イベントに分類されるのは，周期型と不規則型とする場合が多い．不規則型は，周期は変動するがあ

注1：「し」は風とする説もある．

〔図1〕雨の降り方と非周期イベント

イベント到着モデル

周期型(periodic)
　周期の変動は1%程度
不規則型(irregular)
　周期は変動するが予測可能な範囲
制限型(bounded)
　最小間隔または最小/最大間隔がある
バースト型(bursty)
　最小間隔なし，しかしイベント密度あり
無制限型(unbounded)
　最小間隔もイベント密度もない
　統計的な分布関数として表現される

別の視点
散発型(sporadic)
　外部周期と内部周期がある

さらに別の視点
五月雨型　途切れがちに繰り返す
時雨型　ひとしきり続いて終わる
村雨型　突然始まって，突然終わる

〔図2〕ネットワークアクセスパターン

らかじめわかっているか，予測可能な範囲に限られる．変動部分をジッタ(jitter)として扱う場合もある．

非周期的イベントは，残りの制限型，バースト型，無制限型である．制限型は，イベント到着の最小間隔がわかっているものである．最小間隔の出所は，物理的なものや要求仕様，システム上の制限などである．たとえば，キーボード入力イベントなどは，人間が打ち込むということとキーが元に戻る時間などがあるので最小間隔が存在する．

このイベントの処理をハードリアルタイムスケジューリングするときは，最悪の条件を考えて最小間隔を周期とするので悲観的な結果になる．バースト型は，最小間隔はないがイベント密度はわかっているようなものである．たとえば，"1秒間に50件のトランザクション処理をする"というような要求仕様などが該当する．

期間と最大件数が指定されているだけで最小間隔は指定されていないので，複数のイベントが同時に到着する可能性がある．そのためバッファリングが必要になる．バッファサイズはイベント密度と処理時間から算出する．ハードデッドラインを設定されても無理な場合が多く，平均応答時間を短縮するようなスケジューリングポリシーを採用せざるをえなくなる．無制限型は，最小間隔もイベント密度もわからないような場合である．スケジューリングのしようがない．

図2のメール読み込み中のパターンを見ると，大きな周期と小さな周期で構成されているように見える．大きな周期を外部周期(outer period)，小さな周期を内部周期(inner period)と呼ぶ．それぞれの周期がジッタを含んでいて多少の変動があるが，このような大きな二重構造は変わらないものをスポラディックイベントと呼び，ランダムな非周期イベントと区別するアイデアがある[1]．

このパターンは，通信が始まるとパケットが集中して到着し，それがしばらく続いてデータを送り終わるとしばらく静かになるという，ネットワークに特徴的なものである．

始まりは散発的だが，始まってしまうとある程度周期的な動作になる．このような動作を散発的周期イベントと呼び，従来のイベントモデルとは区別するのである．

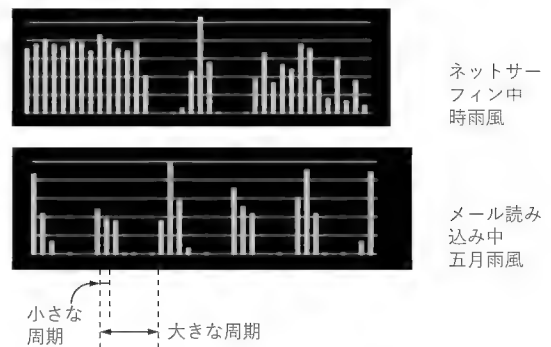
内部周期を最小イベント間隔とすれば従来の制限型に分類されるが，制限型では途切れがちに繰り返す．途切れる部分を考慮しないので悲観的なスケジューリングになってしまう．悲観的とは，実際はスケジュール可能なのに不可能と判定してしまうという意味である．集中している期間が村雨的である場合を，バースト的周期イベントと呼ぶ場合もある．

このバースト期間に受信タスクがビジィになり，受信タスクより優先度の低いタスクが動けなくなるという問題が発生することがある．たとえば，レートモノトニック法などを使ってスケジューリングしていると，キーボード入力のような入力間隔の長い入力を処理するタスクのプライオリティは低く設定されるので，ネットワーク通信が始まるとキー入力を受け付けなくなったりする．これらの問題に対応するのが後述するスポラディックスケジューリングである．

図2の場合の内部周期は，トラフィックを表示するプログラムの表示分解能なので，入力イベントに本質的なものではない．しかし，ある程度の大きさの組み込みシステムでは複数のタスクをパイプ型とかクライアント/サーバ型などのアーキテクチャを使って組み合わせることになる．

この場合，外部からの入力イベントが純粋な周期型であっても一次処理するタスクの実行時間は変動するので，次のタスクへの入力はジッタを含むようになる．変動の原因がバッファサイズなどであるとジッタにある種の周期構造が入ることになる．このようなものが内部周期の原因になることもあるので，表示分解能も十分内部周期の原因になるかもしれない．

2 制限型のスケジューリング

制限型に限らず，デッドラインをもった非周期イベントに対する方法には，割り込みハンドラと処理タスクの組み合せ，ポーリングタスクと処理タスクの組み合せ，周期サーバと処理タスクの組み合せが一般的である．周期サーバの一つとしてスポラディックサーバがある．

- ポーリング方式

イベント入力に割り込みを使用できない場合は，ポーリング方式を使用することになる．これはポーリング用の周期タスク

を立ち上げておき，イベントの有無をモニタする方法である．イベント入力があった場合に処理用タスクを起動する．処理用タスクの起動周期が最小イベント間隔に対応するようになる．それでは，ポーリング用タスクの周期は何から決まるかというと，対象とするイベント処理のデッドラインから決まる．

イベント処理のデッドラインをD，ポーリング周期を$Tpoll$，イベント処理全体のデッドラインを$Dpoll$とすると，

$$D \geqq Tpoll + Dpoll$$

とする必要がある（**図3**）．$Dpoll$は，ポーリングタスクと処理用タスクの最悪実行時間の和ではなく，処理全体のデッドラインである．つまり，$Dpoll$の間にはプライオリティのより高いタスクが走ったり，割り込みが入ったりする．

たとえば，最小イベント間隔が50msの場合でも，このイベントのデッドラインが5msであれば，ポーリングタスクは50msよりもはるかに短い周期でポーリングしなければならない．このためオーバヘッドが無視できなくなる．そのため，デッドラインが厳しい場合にはポーリング方式は向いていない．

この例の場合，イベント処理の最悪実行時間が3msかかるとすれば，その時間を確保するために$Dpoll$は3ms以上にする必要がある．その結果，ポーリング周期は2ms以下にしなければならない．最悪でも50msに1回しか発生しないイベントのために，2msでポーリングするのはかなりの無駄である．

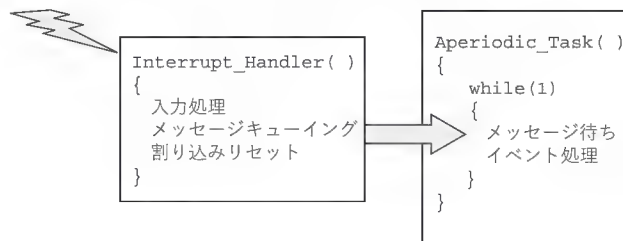
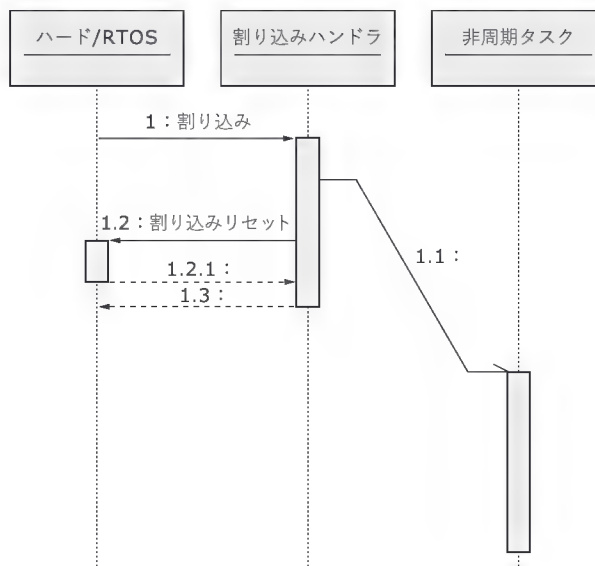
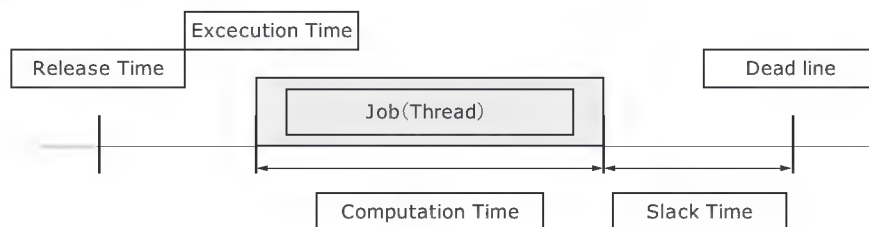
● 割り込みハンドラ方式

イベントの検出に割り込みを使える場合には，ポーリングタスクの代わりに割り込みハンドラを使用する．したがって，余計なオーバヘッドはかからなくなる．**図4**に，一般的な実装方法を示す．イベントが入ると割り込みハンドラが起動され，入力デバイスなどからデータを取り込む．データが整うと，次に処理用のタスクを起動する．

〔図3〕ポーリング周期と最小イベント間隔

〔図4〕制限型イベントの割り込み方式の処理

〔図6〕バースト型イベントの割り込み方式の処理

〔図5〕制限型イベントの割り込みリセットタイミング

〔図7〕バースト型イベントの割り込みリセットタイミング

〔図8〕スラックタイム

最小入力間隔がイベント処理のデッドラインより短い場合の起動は，イベントフラグのようなキューイングしないものを使用する．デッドラインが長い場合，というか処理時間がかかる場合には，処理中に次のイベント入力が入るので，処理用タスクの起動にはキューイングできるセマフォやメールボックスなどを使用する．

また，**図5**は処理用タスクの最後で次の割り込みを解除しているが，処理時間が長い場合にはこの部分も変更する必要がある．つまり，デッドラインが長い場合には**図6**に示したバースト型の場合と同様なパターンになる(**図7**)．

割り込みハンドラ方式の場合は，最小イベント間隔が50msであれば最悪でも50msに1回割り込みが入るだけであるが，何らかの理由で割り込みがもっと頻繁に入ると処理用タスクよりもプライオリティの低いタスクが影響を受けてしまう．

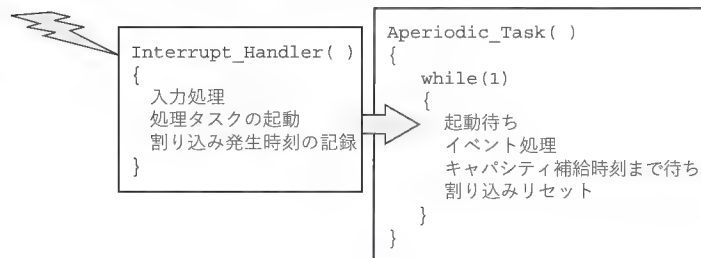
たとえば，デバイスが故障して割り込みをかけっぱなしにするかもしれないし，悪意を持った割り込み攻撃を受けるかもしれない．ネットワークテロリストから組み込みシステムを守るためには，割り込みハンドラ方式では不十分である．

3 スポラディックサーバ

周期サーバ方式には，プライオリティ交換サーバ(Priority Exchange Server)，遅延サーバ(Deferrable Server または Deferred Server)，スポラディックサーバ，スラックスチルサーバ(Slack Stealer)がある．またこれらのサーバの中に，さらに実現方法により何種類かの分類がある[2]．

よく使われるのはスポラディックサーバである．スラックスチルサーバは，デッドラインまでの空き時間(slack time，**図8**)を有効に使おうとする方式で，まだいろいろと論文発表されている比較的新しい方法である．プライオリティ交換サーバと遅延サーバは，1990年頃の技術である．

図9に，もっとも簡単なスポラディックサーバの実装パターンを示す．この方式であれば，ほとんどのRTOSでスポラディックサーバを実現できる．**図4**との違いは，割り込みハンドラ内で時刻を記録する点と，処理タスク内でその時刻を使って起動されたときから一定の時間(補給周期)待ちに入る点である．処理タスクの中で割り込みを許可する前に一定の時間止まってしまうので，非周期イベントの処理をやりすぎてシステム全体

〔図9〕制限型イベントのスポラディックサーバの処理

キャパシティ補給時刻 = 割り込み発生時刻 + 補給周期

に悪い影響が出ないようにしている．

一般的なスポラディックサーバを実現するためには，タスクの実行時間を精密に測定する必要がある．精密というのは，システムクロック程度の精度では粗すぎる場合があるという意味である．普通，システムクロックは，タイムアウト処理や前述のポーリング周期など外部の事象に対して精度を決める．スポラディックサーバで要求されるのはシステム内部の事象を計測するための精度である．

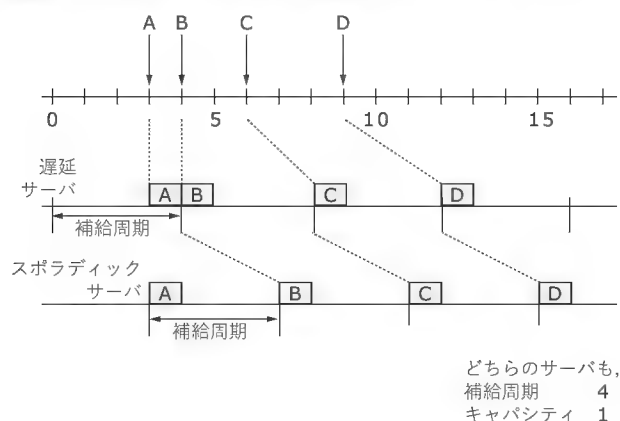
処理用タスクにはあらかじめ使ってもよいCPU時間が与えられている．このCPU時間を容量(capacity あるいは budget)と呼ぶ．このCPU時間を使い切るまではデッドラインモノトニックに割り当てられた処理用タスク本来のプライオリティで実行することが可能だが，使い切るとバックグラウンドプライオリティで細々と実行を続けるかサスペンドされてしまう．そして，補給周期(replenishment period)を経過すると再び元のプライオリティに戻る．**図9**の例は，

　イベント処理応答時間＝補給周期
　イベント処理時間＝容量

としたものである．

補給周期はどこから計測するかというと，処理用タスクが起動された時点から計測を始める．バックグラウンドプライオリティに落とされた時からではない．スポラディックサーバとよく似た遅延サーバでは，タスクが起動された時点ではなく，補給周期のスタート時点から計時する．遅延サーバは，起動を遅延する機能をもった周期タスクと考えられる．

一方，スポラディックサーバは，最初に起こったイベントによって位相が決定される周期タスクのようにふるまう．このため処理タスク起動間隔を確保することができる．したがって，優

〔図10〕スポラディックサーバと遅延サーバ

〔図11〕スポラディックサーバと位相

先度の低いタスクにもCPU時間を割り当てることが可能になる.

図10に示したように，遅延サーバではイベント処理がAとBのように連続してしまう場合があるが，スポラディックサーバでは連続することはない．一度決定された位相は，イベント入力が補給周期以上途切れることでリセットされる．そして時雨のように，次のイベント入力群が始まると新しい位相を固定して間を空けながら処理する(**図11**)．スポラディックサーバは，時雨を五月雨に変換することができる.

処理用タスクは，普通のタスクと同様に扱われるので他のタスクと同様にブロックされたりもする．ブロック後しばらくして実行を再開した場合は，再開した時点を起点として再び補給タイミングを計算する．そして，その時の補給容量はブロックされるまで実行したCPU時間になる．補給するという意味は，残っている容量に補給容量を加えるということである.

また，このときにプライオリティがバックグラウンドになっていたら，通常のプライオリティに戻す．ブロックではなくプリエンプトされた場合には，実行容量を減らすだけで，補給処理をスケジュールすることはしない.

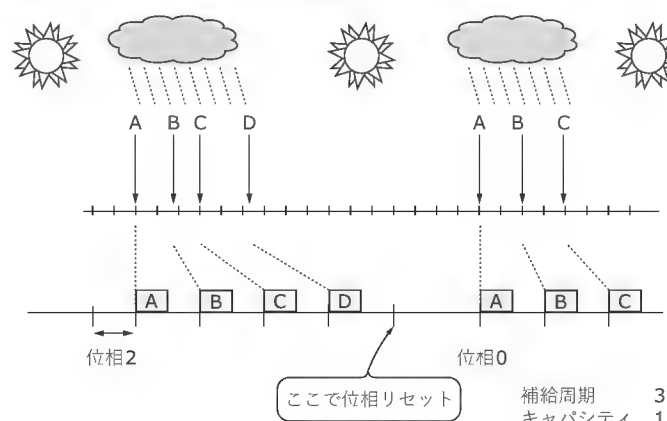
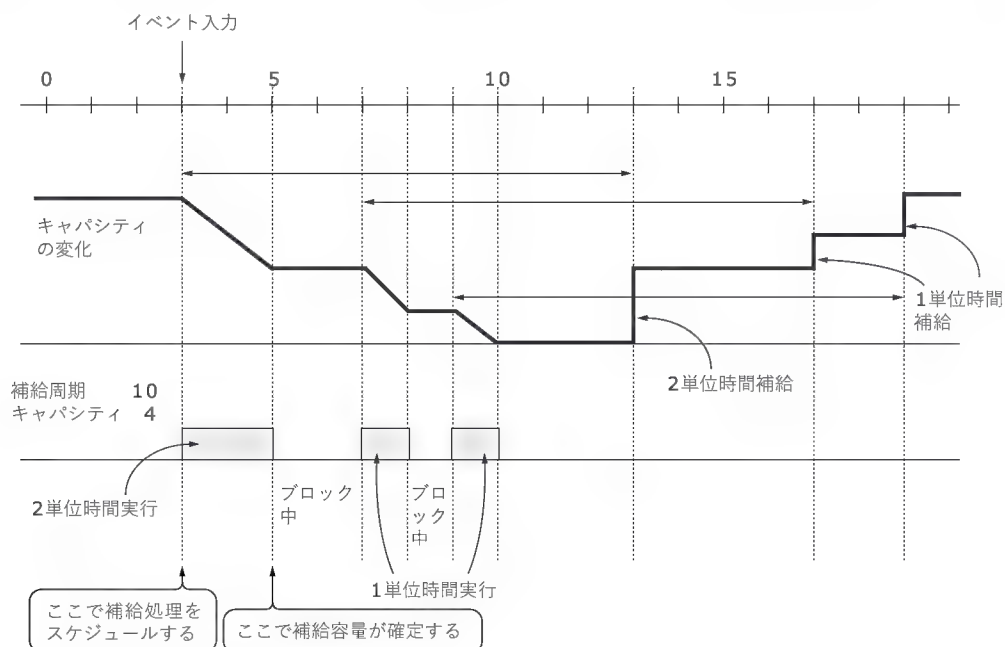
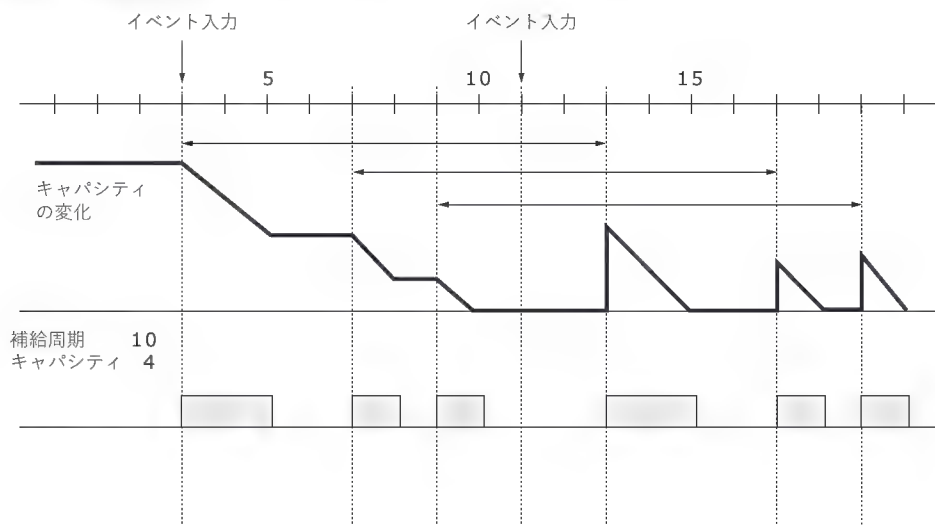



図12の例は，容量4補給周期10の非周期イベントが時刻3で入力された場合を示している．時刻3に処理タスクが起動されて時刻5まで実行したところでブロックされた．起動時(時刻3)に次の容量補給処理は時刻13であることが確定し，ブロック時(時刻5)に補給容量が2であることが確定する．時刻8にブロックした際と時刻10に実行が終了した際にも同様に補給処理がスケジュールされる．時刻11に次のイベントが入力された場合を**図13**に示す.

時刻13までは，容量がないのでバックグラウンド処理となる．この場合，実行されないことにした．時刻13で容量として2単位のCPU時間が補給されるので起動される．起動されるので時刻23が次の補給タイミングになる．時刻15まで実行すると容量

〔図12〕
スポラディックサーバの補給処理
(POSIXの場合)

〔図13〕スポラディックサーバの補給処理-2

がなくなるので，ブロックなどの事象が起きなくとも現在のプライオリティを放棄するので，バックグラウンド処理にまわる．

このように，スポラディックサーバを使用すると，動作パターンを記憶するような形で処理が徐々に五月雨的実行状態に分割されていく．ただし，あまり分割されすぎるとパフォーマンス上の問題があるので，POSIX(IEEE 1003.1d)で仕様化されているスポラディックポリシーでは，補給回数の上限を指定するパラメータが追加されている．

図9の実装では，処理用タスクが途中でブロックされるたびに補給タイミングを計算していない．**図12**に示した仕様どおりに実装するとすれば，アンブロックからブロックまでのそのタスクが消費した正味のCPU時間を計測する必要がある．どの程度連続して走れるかはシステム全体のタスク構成や割り込み頻度によって変わる．

細かく分断されながら実行するタスクの実行時間を求めるためには，かなりの精度で実行時間を測る必要がある．このためには，RTOSのディスパッチャレベルでの対応が必要になる．そのため，効率の良いスポラディックサーバを実装するためにはRTOSのサポートが必要になるといわれている．いままでいわれているだけでほとんどのRTOSメーカーは固定優先度のプリエンプトスケジューラで満足していた．カーネルにはほとんど手を入れずミドルウェアの拡張にベクトルが向いていた．カーネルはミドルウェアを走らせるプラットホームとなった感がある．その挙げ句がRTOSメーカーの減少，パブリックドメインOSの進出などであり，いつまでもこのままではRTOS技術自身の存在が危ういのではないかと思っていたが，いくつかのRTOSはがんばっている．

たとえばQNXは，POSIXのスポラディックサーバをサポートしている．POSIX対応RTOSを使用する場合は，散発的イベントを処理するタスクに対してスポラディックポリシー(SCHED_SPORADIC)を指定するだけで良い．QNXのサンプルソースコードは，

ftp://ftp.embedded.com/pub/2002/12vanzandt/

からダウンロードできる．rtl-sporadic.tgzを解凍してrtl.cを見ると，SCHED_SPORADICで動作するスレッドの作り方がわかる．

ITRONの場合は，オーバランハンドラを使用すれば処理用タスクの残り容量を知ることができるのと，容量を使い切ったときにオーバランハンドラが起動されるので，スポラディックサーバを実装できるかもしれない．

ただし実装依存の部分があるので，スポラディックスケジューリングに使用できるか否かはRTOSごとに確認する必要がある．システムクロックはインターバルタイマの割り込み回数を利用して，オーバランハンドラはインターバルタイマのカウンタ値を利用するような使い分けが必要ではないかと思う．T-Kernelは，ITRON4.0の延長上にあるのかと思ったらITRON3.0をベースとするらしい．

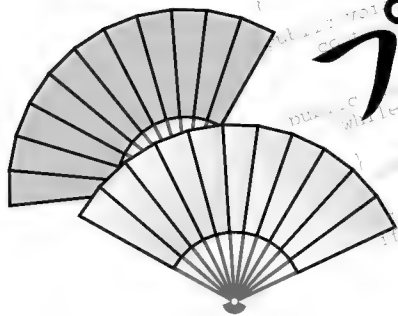
おわりに

スポラディックサーバの動作について説明した．バーストモードの動作とデッドラインが周期よりも長い場合，メッセージ通信などキューイングが発生する場合については別の機会に説明する．

参考文献

1) N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, *Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling*, YCS 120 Department of Computer Science, University of York (February 1992). http://www.cs.utah.edu/~regehr/reading/ からダウンロード可能

2) Liu, Jane W. S., *Real-Time Systems*, Prentice Hall: Upper Saddle River, NJ. (2000)

ふじくら・としゆき 日本ラショナルソフトウェア(株)

宮坂電人

第3回

開放/閉鎖原則(実践編)

今回も前回と同様に,『オブジェクト指向入門』注1を取り上げます.前回は,開放/閉鎖原則が出てくるまでの同書の内容について簡単に紹介しました.いよいよ原則の説明です.同書によれば,

「ある手法がモジュールの分解のしやすさを満足させるために,開放的であると同時に,閉鎖的であるモジュールを作り出さねばならない」

ということです.しかし,これだけでは何のことなのかわかりません.だいいち「開放的であると同時に,閉鎖的である」というのが理解不能です.まったく正反対のことを同時にやれといわれているようなものです.じつは,ここで「開放的」,「閉鎖的」と述べているのは,違った観点での要求です.それは,

- **開放的**:モジュールが拡張可能であるなら「開放的(Open)」と評する(モジュールの実装者側から見た評価)
- **閉鎖的**:モジュールが他のモジュールから利用できるなら「閉鎖的(Closed)」と評する(モジュールの利用者側から見た評価)

ということです.わかりにくいのは,なぜほかのモジュールから利用できることを「閉鎖的」と評するかです.同書によれば,

- モジュールが適切に定義され,インターフェースが安定している状態を,ほかのモジュールから利用できると評する

ということです.モジュールを提供する場合,提供時点のバージョンでフィックスし,ファイルを閉じて(Close)提供します.「これ以上モジュールをいじらないので注2,どうぞ安心して使ってください」というのがClosedにこめられた意味です.

安定したインターフェース

「適切に定義され,インターフェースが安定している」とは,どういうことでしょうか? ここで一つの例として,男女の人数をカウントしていくモジュールをC言語で記述したとします.いろいろな方法がありますが,男の数と女の数を記録する構造体を決めます.また,構造体をクリアしたり,男女の合計人数を得る関数も用意します.**リスト1**,**リスト2**のようになります.

このPersonCounter.hとPersonCounter.cを利用する例は,**リスト3**のようになります.

いうまでもなく,ここで示しているプログラムは厳密な意味でのオブジェクト指向ではなく,開放/閉鎖原則でいうところの「閉鎖的」でもありません.というのも,もしここで仕様変更があり,性別不明の人数も配慮したいとなると,どうなるでしょうか.もっとも安直ですが,ありがちな対応はPersonCounter構造体に性別不明メンバを追加することです.つまり,**リスト4**,**リスト5**のような対応でしょう.これは一見,うまい対応を

〔リスト1〕PersonCounter.h(最初のバージョン)

```
struct PersonCounter {
    int mMale;
    int mFemale;
};

void PC_Reset(struct PersonCounter *); /* 構造体をクリアする */
int PC_Total(struct PersonCounter *);  /* 合計の人数を算出する */
```

〔リスト2〕PersonCounter.c(最初のバージョン)

```
#include "PersonCounter.h"

void PC_Reset(struct PersonCounter *iPC)
{
    iPC->mMale = 0;
    iPC->mFemale = 0;
}

int PC_Total(struct PersonCounter *iPC)
{
    return iPC->mMale + iPC->mFemale;
}
```

〔リスト3〕PersonCounter(最初のバージョン)**の利用例**

```
static void Test()
{
    struct PersonCounter aPC;

    PC_Reset(&aPC);

    aPC.mMale += 10;
    aPC.mFemale += 10;

    printf("total = %d¥n",PC_Total(&aPC));
}
```

注1:バートランド・メイヤー著,アスキー,ISBN4-7561-0050-3.原題はObject-Oriented Software Construction.翻訳書は1990年発行で初版だが,原書はすでにSecond Editionが出ている.そちらはPrentice Hall,ISBN0-13-629155-4.参考URLはhttp://archive.eiffel.com/doc/oosc/.

注2:「いじらない」というのは外部仕様をこれ以上いじらないという意味.バグが発生しても修正しませんという意味ではない.

〔リスト4〕**PersonCounter.h**(性別不明追加)

```
struct PersonCounter {
    int mMale;
    int mFemale;
    int mUnknown; /* (追加) */
};
```

〔リスト5〕**PersonCounter.c**(性別不明追加)

```
void PC_Reset(struct PersonCounter *iPC)
{
    iPC->mMale = 0;
    iPC->mFemale = 0;
    iPC->mUnknown = 0; /* (追加) */
}

int PC_Total(struct PersonCounter *iPC)
{
    return iPC->mMale + iPC->mFemale + iPC->mUnknown; /* (変更) */
}
```

〔リスト6〕**PersonCounter.java**

```
public class PersonCounter {
    protected int mMale;
    protected int mFemale;

    public PersonCounter(){
        reset();
    }
    public void reset(){ //クリアする
        mMale = mFemale = 0;
    }
    public int total(){ //合計の人数を算出する
        return mMale + mFemale;
    }
    public int male(){ //男の人数をえる
        return mMale;
    }
    public int female(){ //女の人数をえる
        return mFemale;
    }
    public void addMale(int iMale){ //男の人数を加算する
        mMale += iMale;
    }
    public void addFemale(int iFemale){ //女の人数を加算する
        mFemale += iFemale;
    }
}
```

〔リスト7〕**PersonCounter.javaの利用例**

```
public class javasample {

    public static void test(){
        PersonCounter aPC = new PersonCounter();

        aPC.addMale(10);
        aPC.addFemale(10);

        System.out.println("total = "+ aPC.total());
    }

    public static void main (String args[]) {
        (new javasample()).test();
    }
}
```

したかのように思えます．ところがPersonCounter構造体をファイルに記録していたり，構造体のサイズがint二つ分の前提でフォーマットを決めていた場合，そちらに悪影響が出ます．また，間違った対応ではあるものの，PC_ResetやPC_Totalを利用せずにメンバを直接操作するコードがあった場合，mUnknownメンバの存在を無視しているためプログラムがまともに動作しなくなります．

PersonCounterを利用していた人にとっては，突然mUnknownメンバという外部仕様が現れたので，ちっとも閉じて(Close)はいなかったわけです．さきほどの開放的，閉鎖的の評価でいえば，

- **開放的の評価**：実装者が勝手にいじれる状態なので開放的(というか無節操)
- **閉鎖的の評価**：モジュールが他のモジュールから安定して利用できないので閉鎖的ではない

ということです．

コピー＆ペーストによる対応

そこで，次にありがちな対応は，PersonCounterはそのままにし，新たにPersonCounterExとでも名づけた別モジュールを作成することです．作成するにあたってPersonCounter.hやPersonCounter.cを複製し，新たにPersonCounterEx.hやPersonCounterEx.cを作り，中身を書き換えるわけです．こうすると，少なくともPersonCounterを利用しているモジュールが実装者の勝手な変更の影響を受ける心配はなくなりそうです．しかし，この方法にも問題点があることが知られています．

- コピー元のモジュールにバグがあった場合，修正が面倒になる
- コピー元のモジュールに仕様追加があった場合，同じような追加作業がコピーした箇所だけ必要になる

といったあたりです．このへんは，毎度おなじみの人海戦術や残業で補えばいい(?!)という反論も出てきそうですが，同じようなことをむなしく繰り返しているような気がします．

開放的，閉鎖的の評価でいえば，

- **開放的の評価**： 実装者が勝手にいじれる状態なので一見開放的なように錯覚する．しかし，修正や仕様追加で苦労する
- **閉鎖的の評価**：モジュールがほかのモジュールから安定して利用できるので閉鎖的である．ただし，実装者の苦労が影響するかもしれない

ということです．

継承を使った対応

PersonCounterをJavaで記述した例を示しましょう．さきほどはPersonCounterを構造体で実装しましたが，当然のことながらクラスで実装し，オブジェクト指向の定石である「内部実装(フィールド変数)を公開しない」にしたがったものにします．

このモジュール(PersonCounter.java：**リスト6**)を利用する例は，**リスト7**のようになります．

ここで，同じように性別不明の対応が別のグループから要求されたとしましょう．もっともまずい対応は，PersonCounter

をいじってしまうことです．これでは，さきほどのC言語の実装で起こった問題を再現するだけです．かといって，コピー＆ペーストで新たにクラスを作成するのも，やはり問題があります．ここでは，継承によってPersonCounterExという子クラスを作って対応しましょう（**リスト8**）．

こうしておいて，別のグループにはPersonCounterExを提供すればよいわけです．性別不明の対応をするためにPersonCounterをいじっていないため，PersonCounterの利用者にとっては，

- **閉鎖的の評価**：モジュールが他のモジュールから安定して利用できるので閉鎖的である

いっぽうモジュールの実装者にとっては，

- **開放的の評価**：実装者はPersonCounterの機能拡張であるPersonCounterExを提供できるので開放的

という評価です．継承は差分の記述であってコピー＆ペーストではないので，

- 継承元のモジュールにバグがあった場合，継承元の修正だけですむ（ただし，その修正の影響によって子クラスの修正が発生する場合もあり得る）
- 継承元のモジュールに仕様追加があった場合，追加作業は継承元だけにとどまるので，子クラスへコピー＆ペーストしなくてよい（ただし，追加の影響によって子クラスの修正が発生する場合もあり得る）

というメリットもあります．

PersonCounterの情報をファイルに記録していた場合はどうなるでしょうか？　男女の人数を直接mMale，mFemaleでアクセスせず，male()，female()，addMale(...)，addFemale(...)のメソッドだけで行うようにしていたなら，PersonCounterExに移し替えるのも，やはりメソッドだけを使えばよいので，これも問題なしです〔具体的にはmale()，female()で男女の数を保存していたものを，addMale(...)，addFemale(...)で復元する〕．継承を上手に利用にしてモジュールの拡張（というか増設）をするかぎりは，開放/閉鎖原則を守れるわけです．

〔**リスト8**〕**PersonCounterEx.java**

```
import PersonCounter;

public class PersonCounterEx extends PersonCounter {
    protected int mUnknown;

    public PersonCounterEx(){
        reset();
    }
    public void reset(){ //クリアする
        super.reset();
        mUnknown = 0;
    }
    public int total(){ //合計の人数を算出する
        return super.total() + mUnknown;
    }
    public int unknown(){ //性別不明の人数をえる
        return mUnknown;
    }
    public void addUnknown(int iUnknown){ //性別不明の人数を加算する
        mUnknown += iUnknown;
    }
}
```

開放/閉鎖原則は理念

以上の点から，継承によって開放/閉鎖原則が守れるので，仕様変更/追加が発生したなら，元のクラスをいじらずにどんどん継承でつぎ足しましょうという話に展開しそうですが，これはこれで問題が起こります．というのも，無節操な継承や継承レベルが深いことで起こる障害も報告されるようになったからです．継承は便利な「パッチ」手段ではありません．それに，開放/閉鎖原則とは，「どんどん継承を使いましょう」という意味ではないからです．

たしかに，『オブジェクト指向入門』には，古典的な設計アプローチやプログラミング方法で開かれていると同時に閉じているモジュールを書くことは不可能で，この表面上のジレンマを解決できるのは継承だけだ，という記述があります．しかし，原則というのは「こうあるべき」という“理念”であって，「このように作るべき」という“実装”ではありません．理念と実装の混同をしてはいけない，ということです．

「ジレンマを解決できるのは継承だけだ」と述べているのは，実装面での観点です．そこでここからは，実装面で継承をどう利用すれば，理念としての開放/閉鎖原則を保証できるのかを検討してみましょう．

開放/閉鎖原則とデザインパターン

『オブジェクト指向入門』を書かれた時点では，まだデザインパターンはさほど話題になっていませんでした．そのせいかどうかはわかりませんが，同書にはデザインパターンを使って開放/閉鎖原則を保証するというトピックは載っていませんでした．そこで僭越（?!）ながら，GoF本[注3]にあるデザインパターンで，開放/閉鎖原則を実現するパターンを検討してみましょう．これは人によって意見が分かれるところですが，

- **開放**：モジュールの実装者にとってモジュールを拡張しやすい
- **閉鎖**：安定しているので，利用者にとってモジュールを利用しやすい

という見地で探してみると，

- **Template Method**：処理のテンプレートを用意し，実際の処理はサブクラスにまかせる
- **Command**：要求をカプセル化し，undo可能にする
- **Iterator**：イタレータを実現する
- **State**：内部状態が変化したときに，ふるまいを変えられるようにする

注3：『オブジェクト指向における再利用のためのデザインパターン』改訂版，ソフトバンク，ISBN4-7973-1112-6

- **Strategy**：アルゴリズムを交換可能にする
- **Visitor**：オブジェクト内の要素の巡回操作を導入しやすくする

あたりでしょう．ほかのパターンでも開放/閉鎖原則を実現する工夫はあるかもしれませんが，簡単に見つかる範囲ではこうだということです．

では，それぞれのパターンでどのように開放/閉鎖原則を保証しているかどうかを検討してみましょう．

Template Methodでの開放/閉鎖原則

Template Methodは，操作の外部仕様を明示するベースクラスを用意し，実際に操作をどのように実装するかは，ベースクラスを継承したクラスで決めてしまうパターンです．実装者はベースクラスの仕様を利用者に提示しておき，利用者が使うのはベースクラスのメソッドのみを使うように要請するわけです．

実装者は，提示した外部仕様を保証できるかぎりサブクラスを自由に実装できます（実装者にとって開放的）．また，サブクラスがどういじられようとも提示した外部仕様が保証されているかぎり，利用者は安定してクラスを利用できます（利用者にとって閉鎖的）．

Commandでの開放/閉鎖原則

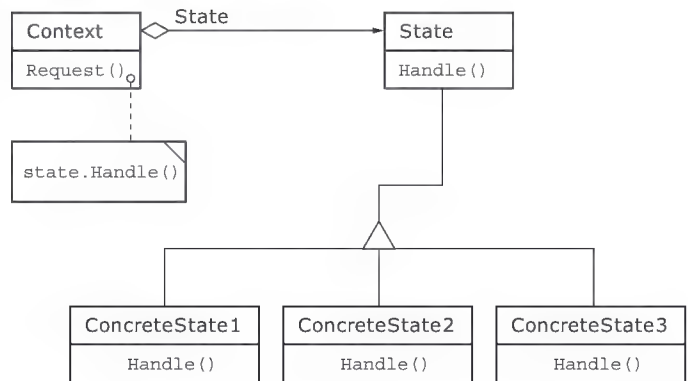
Commandは，要求をオブジェクトとしてカプセル化することで，要求そのものを記録したり複合化したり取り消し可能にできるパターンです．実装者側は，コマンドのベースクラスを利用者に提示しておき，利用者が使うのはカプセル化されたコマンドのオブジェクトのみとします．

実装者は，新しいコマンドを追加/拡張できます（実装者にとって開放的）．また利用者は，コマンドがどのように実装されているか気にしなくてよく，コマンドの追加/拡張を気にせず安定してクラスを利用できます（利用者にとって閉鎖的）．

Iteratorでの開放/閉鎖原則

Iteratorは，コンテナ（複数のオブジェクト/データを保持したオブジェクト．GoF本では「集約オブジェクト」と表現している）を巡回するイタレータを実現するパターンです．

コンテナ内の巡回というのは，やっかいな問題です．どのようにコンテナを実装しているかを利用者に教えてしまうと，コンテナの内部実装に依存したコードを利用者が書いてしまいます．その後でバグが見つかったり，あるいはパフォーマンス向上や仕様変更などでコンテナの内部実装を変更してしまうと，変更前の内部実装に依存したコードを書いた利用者が困ってしまいます．

内部実装を公開せずにイタレータを提供すると，あとで内部実装を変更しても，その影響はイタレータの実装を変更するだけですみます．つまりイタレータの変更という手間を実装者が引き受けるだけで実装者は好きなように内部実装をいじることができます（実装者にとって開放的）．また，利用者はどのように内部実装がされているか気にしなくてすみますし，内部実装がどう変更されようと気にせずに安定してクラスを利用できます（利用者にとって閉鎖的）．

〔図1〕Stateパターン

〔図2〕Strategyパターン

Stateでの開放/閉鎖原則

Stateは，オブジェクトが内部状態を変化したときにオブジェクトのふるまいを変えるようにするパターンです．内部状態がどうであろうと，利用者にはベースクラス（**図1**の`State`クラス）を提示しておけば，仕様変更/追加によってオブジェクトの内部状態に新しいものが増えたり，状態の変化の仕方が変わっても，影響を受けるのは実装者であり，利用者ではありません（利用者にとって閉鎖的）．また，利用者に迷惑をかけずに新しい状態を追加したり，状態の変化の仕方を変えられます（実装者にとって開放的）．

Strategyでの開放/閉鎖原則

Strategyはアルゴリズムを交換可能にすることで，最適なアルゴリズムを選択できるようにするパターンです．アルゴリズムをどう実装しようと利用者にはベースクラス（**図2**の`Strategy`

〔リスト9〕**container.java**（まずい実装例）

```
public class container {

    public Object[] mArray;
    public int mRegistCount;
    private final int ExpandCount = 1024;

    public container(){
        mRegistCount = 0;
        mArray = new Object[ExpandCount];
    }

    public void append(Object iObj){ //オブジェクトを追加する
        int aArrayLength = mArray.length;
        if(mRegistCount >= aArrayLength){
            Object[] aNewArray = new Object[aArrayLength + ExpandCount];
            for(int aIndex = 0; aIndex < mRegistCount; aIndex++){
                aNewArray[aIndex] = mArray[aIndex];
            }
            mArray = aNewArray;
        }
        pushTop(iObj);
        ++mRegistCount;
    }

    public Object search(Object iObj){ //オブジェクトを検索する
        for(int aIndex = 0; aIndex < mRegistCount; aIndex++){
            if(iObj.equals(mArray[aIndex])){
                Object aObj = mArray[aIndex];
                while(aIndex < mRegistCount){
                    mArray[aIndex] = mArray[aIndex + 1];
                    ++aIndex;
                }
                pushTop(aObj);
                return aObj;
            }
        }
        return null;
    }

    private void pushTop(Object iObj){
        for(int aIndex = mRegistCount; aIndex > 0; aIndex--){
            mArray[aIndex] = mArray[aIndex - 1];
        }
        mArray[0] = iObj;
    }
}
```

〔リスト10〕**container.java の利用者側のコード例**

```
public class javasample {
    ...(略)...
    private void dumpContainer(container iCont){
        for(int aInt = 0; aInt < iCont.mRegistCount; aInt++){
            System.out.print("(" + iCont.mArray[aInt] + ")");
        }
        System.out.println();
    }

    public void test2(){
        int aInt;
        container aCont = new container();
        for(aInt = 1; aInt <= 7; aInt++){
            aCont.append(new Integer(aInt));
        }
        dumpContainer(aCont);
        for(aInt = 0; aInt < 10; aInt += 3){
            Object aObj = aCont.search(new Integer(aInt));
            System.out.println("aInt = " + aInt + ",aObj = " + aObj);
            dumpContainer(aCont);
        }
    }
    ...(略)...
}
```

クラス）を提示しておけば，アルゴリズムの変更や追加による影響を利用者に与えなくてすみます（利用者にとって閉鎖的）．

また，利用者に迷惑をかけずに新しいアルゴリズムを追加できます（実装者にとって開放的）．

Visitorでの開放/閉鎖原則

Visitorは，オブジェクト構造上の要素で実行する操作を導入しやすくするために利用するパターンです．Iteratorパターンのときと同様に，あるオブジェクト内の要素がどのように実装されているかを利用者に教えてしまうと，オブジェクトの内部実装に依存したコードを利用者が書いてしまいます．その後にバグが見つかったり，あるいはパフォーマンス向上や仕様変更などでオブジェクトの内部実装を変更してしまうと，変更前の内部実装に依存したコードを書いた利用者が困ってしまいます．

内部実装を公開せずにVisitorパターンによる“突破口”を提供すると，あとで内部実装を変更しても，その影響は突破口の実装を変更するだけですみます．つまり，突破口の変更という手間を実装者が引き受けるだけで，実装者は好きなように内部実装をいじることができます（実装者にとって開放的）．また，利用者はどのように内部実装がされているか気にしなくてすむし，内部実装がどう変更されようと気にせずに安定してクラスを利用できます（利用者にとって閉鎖的）．

開放/閉鎖原則とデザインパターンの実例

デザインパターンの実例を示すことで，開放/閉鎖原則とデザインパターンの関連性を見せたいところですが，ページ数の都合もあるので，ここではIteratorパターンの実例だけを示します．ここで例題として「オブジェクトを登録し，先頭から検索するコンテナ」を実装してみます．ただし，検索したオブジェクトを先頭に配置することで次回の検索で，そのオブジェクトが検索しやすいように実装してみます．

まず，まずい実装例を**リスト9**に示します．

これはいろいろな意味でまずいコードなのですが，まず目につくのはmArray，mRegistCountという内部実装をそのまま外部に公開している点です．containerを利用しているコード（利用者側）は**リスト10**のようになっています．

どうやらdumpContainerメソッド内でcontainerの内部実装であるmArray，mRegistCountを直接利用しているようです．いうまでもなく，これではcontainerの内部実装を変更されてしまうとdumpContainerメソ

〔リスト11〕container.java(Iteratorを導入)

```
import java.util.*;

public class container implements Iterator {
//(追加)implements Iterator

    protected Object[] mArray;  //(変更)public→protected
    protected int mRegistCount; //(変更)public→protected
    private final int ExpandCount = 1024;
    ...(略)...
    //(iterator機能追加)
    protected container mRef = null;
    protected int mIndex;

    private container(container iRef){
        mRef = iRef;
        mIndex = 0;
    }

    public Iterator getIterator(){
        return new container(this);
    }

    public boolean hasNext(){
        return (mRef != null && mIndex < mRef.mRegistCount);
    }

    public Object next(){
        if(hasNext()){
            return mRef.mArray[mIndex++];
        }
        return null;
    }

    public void remove(){
        throw new UnsupportedOperationException();
    }
}
```

ッドの中身を書き換えざるをえません(利用者にとって閉鎖的ではない).いずれにせよ,containerがきわめて効率の悪い実装がされていることがわかっていても,うかつに書き換えられないわけです.

次に,まずい実装はそのままにして,containerにIteratorを導入してみましょう.**リスト11**のようになります.

内部実装はprotectedになったので,利用者側のコードも影響を受けます(**リスト12**).

見てわかるとおり,完全に内部実装に依存するコードでなくなっています.さて,ここでcontainerをもっと効率の良い実装に交換しましょう.いろんな実装がありますが,一つの例としてLinkedListを利用した例を**リスト13**に示します.

かなり見通しがよくなりました.さきほどIteratorを導入させた利用者側のコードは,この改変によって影響を受けるでしょうか.まったく影響を受けないことがわかるかと思います.Iteratorを導入することで,実装者にとってはコード改変がしやすくなり(この点は開放的),利用者は安心してcontainerを利用できる(この点は閉鎖的)ようになったわけです.

おわりに

今回の要点を簡単に述べれば,

- 開放/閉鎖原則とは「モジュールは拡張可能である(開放的)と同時に,ほかのモジュールから安定して利用できる(閉鎖的)こと」である
- 開放/閉鎖原則を満足するためには継承を上手に利用すること
- 開放/閉鎖原則を満足するためにデザインパターン(GoFパターン)が利用できる

ということです.最後のデザインパターンに関しては異論があるかもしれません.あくまで筆者の個人的意見ですし,だいいち『オブジェクト指向入門』にはデザインパターンの話が出てこないことを付け加えておきます.

〔リスト12〕Iterator導入後の利用者側のコード

```
public class javasample {
    ...(略)...
    private void dumpContainer(container iCont){
        Iterator aItr = iCont.getIterator();
        while(aItr.hasNext()){
            System.out.print("(" + aItr.next() + ")");
        }
        System.out.println();
    }
    ...(略)...
}
```

〔リスト13〕container.java(LinkedListを導入)

```
import java.util.*;

public class container {

    protected LinkedList mList;

    public container(){
        mList = new LinkedList();
    }

    public void append(Object iObj){ //オブジェクトを追加する
        mList.addFirst(iObj);
    }

    public Object search(Object iObj){ //オブジェクトを検索する
        int aIndex = mList.indexOf(iObj);
        if(aIndex == 0){
            return mList.getFirst();
        }else if(aIndex > 0){
            Object aObj = mList.remove(aIndex);
            mList.addFirst(aObj);
            return aObj;
        }
        return null;
    }

    public Iterator getIterator(){
        return mList.listIterator(0);
    }
}
```

みやさか・でんと miyadent@anet.ne.jp

CQ RISC評価キット

SH-4PCI with Linux 活用研究3

Microwindowsを使った組み込み向けGUIプログラムの作成事例(基礎編)

酒匂信尋

CQ RISC評価キット/SH-4PCI with Linuxのボードには，ハイレゾリューション対応のVGAコントローラが搭載されている．VGAだけでなくグラフィクスも表示可能なハードウェアを活用しない手はない．しかし，SH-4CPUボードに実装されているローカルメモリは16Mバイトと少ない．そこで，少ないメモリでも動作するMicrowindowsを使って組み込み機器向けGUIプログラムを作成する事例を紹介する．（編集部）

はじめに

Linuxが使われている組み込み機器としては通信機器などが多いと思いますが，このような場合，機器自体にマン-マシンインターフェースをもつ必要はありません．内部にWebサーバを走らせておけば，外部からの操作はWWWブラウザ経由で機器の状態の把握から各種設定まで可能です．

では，スタンドアロンの機器の場合はどうすればよいでしょうか．画面表示機能と大容量のディスクとメモリがあれば，Linuxで標準的に使われているGUIとしてX Window Systemを組み込めば良いのですが，ハードウェア資源が厳しい組み込み機器の場合はどうでしょうか．簡易的に実現するのであれば，シリアルポートを実装してパソコンと接続し，ターミナルからログインしてそこから操作することも可能です．しかし扱う人すべてに，コマンドライン操作を強制するのは難しい場合があります．

最近では，組み込み機器でもグラフィクスの表示が可能なLCDを装備しているものも多くなってきました．ユーザーインターフェースとしてグラフィクスを使えれば，初心者でも直感的な操作が可能になります．

そこで，少ないハードウェア資源でも動作可能なウィンドウシステムであるMicrowindows(nano-X)を動作させ，組み込み機器用マン-マシンインターフェースのサンプルプログラムを作成してみます．

1 CQ RISC評価キット/SH-4PCI with Linuxでの開発環境整備

この評価キットには，すでにMicrowindowsが用意されているので，インストール方法については省略します．Microwindowsについては，次のサイトから入手可能です．

```
http://www.microwindows.org/
```

● コンソールの増設

ディスプレイやキーボード，マウスがデバッグの対象になってしまうので，実質的にはコンソールが使えなくなります．そこで，シリアルポートからログインできるようにしてみます．テストプログラムの強制終了などにも便利です．

シリアルのコンソールを追加する場合は，`/etc/inittab`の最後に次の行を追加します．

```
::respawn:/sbin/getty 9600 ttyS0
```

● Makefileとパスの設定

まず，`Makefile`を作成します．プログラムをデモプログラムが収録されている`microwindows/src/demos`に作成することにし，デモプログラムの`Makefile`を流用します(**リスト1**)．

`Makefile`に対応したパスの設定は，**リスト2**のようになります．

2 作成するプログラムの概要

● ウィンドウの表示内容

作成するプログラムは，320 × 240ドットのウィンドウを作成し，その中にウィンドウのタイトル「CQ SH-4/PCI nano-X Sample program」を表示し，四つの操作用ボタンと二つの表示用Windowを作成します．四つのボタンは，処理開始用のStartボタンとStopボタン，そして本サンプルプログラムでは用途未定の二つのボタンとします．

● ボタンの操作

図1と**図2**に，作成したウィンドウプログラムの画面を示します．プログラム起動時はStartボタンと表示用ウィンドウを表示します(**図1**)．Startボタンを押すと，Stopボタンと用途未定の二つのボタンを表示します(**図2**)．表示用ウィンドウにはnano-Xからのイベントの数を表示します．Stopボタンを押すと初期画面に戻ります．

処理内容としては非常にシンプルなサンプルですが，実際に組み込み機器のマン-マシンインターフェースのテンプレートとして使用できるように，用途未定ボタンであっても，ボタンを押されたことに対応するプログラムの定義もします．

3 メインルーチン

nano-Xに限らずGUIのプログラムは，キーボード，マウスなどの操作による状態変化をウィンドウサーバから受け取り，

〔リスト1〕デモプログラムのMakefile

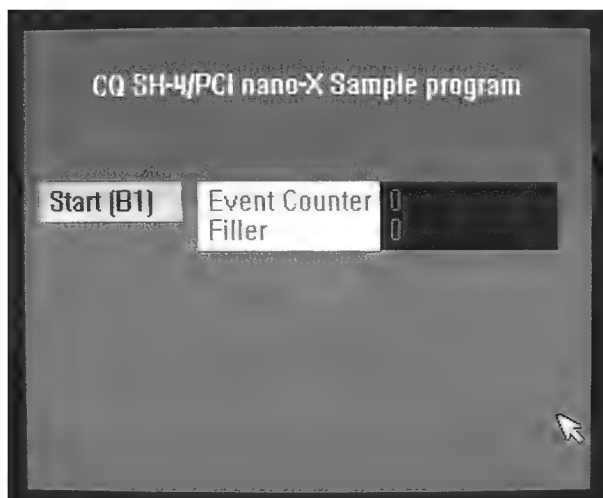
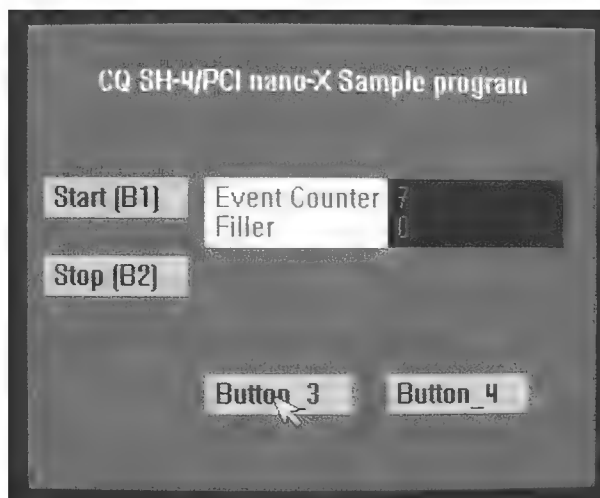
```
##############################################################################
# Microwindows template Makefile
# Copyright (c) 2000 Martin Jolicoeur, Greg Haerr
##############################################################################

include $(CONFIG)

######################## Additional Flags section ############################

# Directories list for header files
INCLUDEDIRS +=
# Defines for preprocessor
DEFINES += -DMWIN

# Compilation flags for C files OTHER than include directories
CFLAGS +=
# Preprocessor flags OTHER than defines
CPPFLAGS +=
# Linking flags
LDFLAGS +=

############################# targets section ################################

ifeq ($(NWIDGET), Y)
ifeq ($(NANOXDEMO), Y)

# If you want to create a library with the objects files, define the name here
LIBNAME =

# List of objects to compile
OBJS = cqsample.o

all: default $(TOP)/bin/cqsample

endif
endif

######################## Makefile.rules section #############################

include $(TOP)/Makefile.rules

######################## Tools targets section ###############################

$(TOP)/bin/cqsample: $(OBJS) $(NANOXCLIENTLIBS) $(TOP)/config
	$(CC) $(CFLAGS) $(LDFLAGS) $(OBJS) -o $@ $(NANOXCLIENTLIBS)
```

その内容に対応した処理を行います．ソースプログラムcqsample.cを**リスト3(a)**に，cqsample.hを**リスト3(b)**に示します．

Grで始まる関数はnano-XのAPIです．詳細については参考

〔リスト2〕Makefileに対応したパスの設定

```
export LD_LIBRARY_PATH=/opt/Emdedix/lwiz1.0/usr/lib
export PYTHONHOME=/opt/Emdedix/lwiz1.0/usr
export PATH=/opt/Embedix/tools/bin:$PATH
export TOP=/opt/lineo-BDK/KMC-BDK/src/microwin/src
export CONFIG=/opt/lineo-BDK/KMC-BDK/src/microwin/src/config
export CC=sh4-linux-gcc
```

〔図1〕起動時の表示状態

〔図2〕Startボタンを押した後の表示状態

〔リスト3〕作成したサンプルプログラム

```
/**************************************************************/
/*                                                            */
/*  Cq SH4/PCI Sample Program                                 */
/*                                                            */
/**************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <ctype.h>
#include <errno.h>
#include <sys/time.h>

#define MWINCLUDECOLORS
#include <nano-X.h>

#include "cqsample.h"


void draw_text(cstate *state)
{
    char buf[32];

    GrFillRect(state->window_02, state->win_02_gcb, 0, 0,
                            WINDOW_02_WIDTH, WINDOW_02_HEIGHT);

    GrText(state->window_02, state->win_02_gcf, TEXT_X_POSITION,
                      TEXT_Y_POSITION, "Event Counter", 13, 0);
    GrText(state->window_02, state->win_02_gcf, TEXT_X_POSITION,
                             TEXT2_Y_POSITION, "Filler", 6, 0);

    GrFillRect(state->window_01, state->win_01_gcb, 0, 0,
                            WINDOW_01_WIDTH, WINDOW_01_HEIGHT);
    sprintf(buf, "%d", state->ev_cnt);
    GrText(state->window_01, state->win_01_gcf, TEXT_X_POSITION,
                        TEXT_Y_POSITION, buf, strlen(buf), 0);
    sprintf(buf, "%d", state->ev2_cnt);
    GrText(state->window_01, state->win_01_gcf, TEXT_X_POSITION,
                       TEXT2_Y_POSITION, buf, strlen(buf), 0);
}

void draw_01_button(cstate *state)
{
    GrFillRect(state->button_01, state->buttongcb, 0, 0,
                            BUTTON_01_WIDTH, BUTTON_01_HEIGHT);
    GrText(state->button_01, state->buttongcf, TEXT_X_POSITION,
                        TEXT_Y_POSITION, "Start (B1)", 10, 0);
}

void draw_02_button(cstate *state)
{
    if(!state->running_buttons_mapped) return;
    GrFillRect(state->button_02, state->buttongcb, 0, 0,
                            BUTTON_02_WIDTH, BUTTON_02_HEIGHT);
    GrText(state->button_02, state->buttongcf, TEXT_X_POSITION,
                          TEXT_Y_POSITION, "Stop (B2)", 9, 0);
}

void draw_03_button(cstate *state)
{
    if(!state->running_buttons_mapped) return;
    GrFillRect(state->button_03, state->buttongcb, 0, 0,
                            BUTTON_03_WIDTH, BUTTON_03_HEIGHT);
    GrText(state->button_03, state->buttongcf, TEXT_X_POSITION,
                            TEXT_Y_POSITION, "Button_3", 8, 0);
}

void draw_04_button(cstate *state)
{
    if(!state->running_buttons_mapped) return;
    GrFillRect(state->button_04, state->buttongcb, 0, 0,
                            BUTTON_04_WIDTH, BUTTON_04_HEIGHT);
    GrText(state->button_04, state->buttongcf, TEXT_X_POSITION,
                            TEXT_Y_POSITION, "Button_4", 8, 0);
}

void job_02_button(cstate *state)
{
    fprintf(stderr, "job 02 ¥n");
    state->state = STATE_STOP;
    start_init(state);
}
void job_03_button(cstate *state)
{
    fprintf(stderr, "job 03 ¥n");
}
void job_04_button(cstate *state)
{
    fprintf(stderr, "job 04 ¥n");
}

void handle_exposure_event(cstate *state)
{
    GR_EVENT_EXPOSURE *event = &state->event.exposure;

    if(event->wid == state->button_01) {
         draw_01_button(state);
         return;
    }
    if(event->wid == state->button_02) {
         draw_02_button(state);
         return;
    }
    if(event->wid == state->button_03) {
         draw_03_button(state);
         return;
    }
    if(event->wid == state->button_04) {
         draw_04_button(state);
         return;
    }
}

void handle_mouse_event(cstate *state)
{
    GR_EVENT_MOUSE *event = &state->event.mouse;

    if(event->wid == state->button_01) {
         state->state = STATE_START;
         start_init(state);
         return;
    }
    if(event->wid == state->button_02) {
         job_02_button(state);
         return;
    }
    if(event->wid == state->button_03) {
         job_03_button(state);
         return;
    }
    if(event->wid == state->button_04) {
         job_04_button(state);
         return;
    }
}

void handle_keyboard_event(cstate *state)
{
    GR_EVENT_KEYSTROKE *event = &state->event.keystroke;
    char c = toupper(event->ch);

    switch(c) {
         case 'Q':
              state->state = STATE_EXIT;
              return;
         case 'N':
              state->state = STATE_START;
              return;
    }

    if(state->state == STATE_STOP) return;

    switch(c) {
         case '2':
              job_02_button(state);
              break;
         case '3':
              job_03_button(state);
              break;
         case '4':
              job_04_button(state);
```

(a) cqsample.c

〔リスト3〕作成したサンプルプログラム(つづき)

```
            break;
    }
}

void handle_event(cstate *state)
{
    switch(state->event.type) {
        case GR_EVENT_TYPE_EXPOSURE:
            handle_exposure_event(state);
            break;
        case GR_EVENT_TYPE_BUTTON_DOWN:
            handle_mouse_event(state);
            break;
        case GR_EVENT_TYPE_KEY_DOWN:
            handle_keyboard_event(state);
            break;
        case GR_EVENT_TYPE_CLOSE_REQ:
            state->state = STATE_EXIT;
            break;
        case GR_EVENT_TYPE_TIMEOUT:
            break;
        default:
            fprintf(stderr, "Unhandled event type %d¥n",
                                            state->event.type);
            break;
    }
}

void start_init(cstate *state)
{
    state->ev_cnt = 0;
    state->ev2_cnt = 0;
    state->level = 0;
    draw_text(state);
    if((state->running_buttons_mapped) && (state->state
                                              == STATE_STOP)){
        GrUnmapWindow(state->button_02);
        GrUnmapWindow(state->button_03);
        GrUnmapWindow(state->button_04);
        state->running_buttons_mapped = 0;
        return;
    }
    if((!state->running_buttons_mapped) && (state->state
                                             == STATE_START)){
        GrMapWindow(state->button_02);
        GrMapWindow(state->button_03);
        GrMapWindow(state->button_04);
        state->running_buttons_mapped = 1;
        return;
    }
}

void init_gui(cstate *state)
{

    state->main_window = GrNewWindow(GR_ROOT_WINDOW_ID,
                        MAIN_WINDOW_X_POSITION,
                        MAIN_WINDOW_Y_POSITION,
                        MAIN_WINDOW_WIDTH,
                        MAIN_WINDOW_HEIGHT, 0,
                        MAIN_WINDOW_BACKGROUND_COLOUR, 0);
    GrSelectEvents(state->main_window, GR_EVENT_MASK_EXPOSURE |
                        GR_EVENT_MASK_CLOSE_REQ |
                        GR_EVENT_MASK_KEY_DOWN |
                        GR_EVENT_MASK_TIMEOUT);

    state->window_01 = GrNewWindow(state->main_window,
                        WINDOW_01_X_POSITION,
                        WINDOW_01_Y_POSITION,
                        WINDOW_01_WIDTH,
                        WINDOW_01_HEIGHT, 0,
                        WINDOW_01_BACKGROUND_COLOUR, 0);
    GrSelectEvents(state->window_01, GR_EVENT_MASK_EXPOSURE);
    GrMapWindow(state->window_01);
    state->win_01_gcf = GrNewGC();
    GrSetGCForeground(state->win_01_gcf,
                                    WINDOW_01_FOREGROUND_COLOUR);
    GrSetGCBackground(state->win_01_gcf,
                                    WINDOW_01_BACKGROUND_COLOUR);

    state->win_01_gcb = GrNewGC();
    GrSetGCForeground(state->win_01_gcb,
                                    WINDOW_01_BACKGROUND_COLOUR);

    state->window_02 = GrNewWindow(state->main_window,
                        WINDOW_02_X_POSITION,
                        WINDOW_02_Y_POSITION,
                        WINDOW_02_WIDTH,
                        WINDOW_02_HEIGHT, 0,
                        WINDOW_02_BACKGROUND_COLOUR, 0);
    GrSelectEvents(state->window_02, GR_EVENT_MASK_EXPOSURE);
    GrMapWindow(state->window_02);
    state->win_02_gcf = GrNewGC();
    GrSetGCForeground(state->win_02_gcf,
                                    WINDOW_02_FOREGROUND_COLOUR);
    GrSetGCBackground(state->win_02_gcf,
                                    WINDOW_02_BACKGROUND_COLOUR);

    state->win_02_gcb = GrNewGC();
    GrSetGCForeground(state->win_02_gcb,
                                    WINDOW_02_BACKGROUND_COLOUR);

    state->button_01 = GrNewWindow(state->main_window,
                        BUTTON_01_X_POSITION,
                        BUTTON_01_Y_POSITION,
                        BUTTON_01_WIDTH,
                        BUTTON_01_HEIGHT, 0,
                        BUTTON_BACKGROUND_COLOUR, 0);
    GrSelectEvents(state->button_01, GR_EVENT_MASK_EXPOSURE |
                                    GR_EVENT_MASK_BUTTON_DOWN);
    GrMapWindow(state->button_01);
    state->buttongcf = GrNewGC();
    GrSetGCForeground(state->buttongcf,
                                    BUTTON_FOREGROUND_COLOUR);
    GrSetGCBackground(state->buttongcf,
                                    BUTTON_BACKGROUND_COLOUR);
    state->buttongcb = GrNewGC();
    GrSetGCForeground(state->buttongcb,
                                    BUTTON_BACKGROUND_COLOUR);

    state->button_02 = GrNewWindow(state->main_window,
                        BUTTON_02_X_POSITION,
                        BUTTON_02_Y_POSITION,
                        BUTTON_02_WIDTH,
                        BUTTON_02_HEIGHT, 0,
                        BUTTON_BACKGROUND_COLOUR,
                        0);
    GrSelectEvents(state->button_02, GR_EVENT_MASK_EXPOSURE |
                                    GR_EVENT_MASK_BUTTON_DOWN);

    state->button_03 = GrNewWindow(state->main_window,
                        BUTTON_03_X_POSITION,
                        BUTTON_03_Y_POSITION,
                        BUTTON_03_WIDTH,
                        BUTTON_03_HEIGHT, 0,
                        BUTTON_BACKGROUND_COLOUR,
                        0);
    GrSelectEvents(state->button_03, GR_EVENT_MASK_EXPOSURE |
                                    GR_EVENT_MASK_BUTTON_DOWN);

    state->button_04 = GrNewWindow(state->main_window,
                        BUTTON_04_X_POSITION,
                        BUTTON_04_Y_POSITION,
                        BUTTON_04_WIDTH,
                        BUTTON_04_HEIGHT, 0,
                        BUTTON_BACKGROUND_COLOUR,
                        0);
    GrSelectEvents(state->button_04, GR_EVENT_MASK_EXPOSURE |
                                    GR_EVENT_MASK_BUTTON_DOWN);
/*
*/
    GrMapWindow(state->main_window);
    state->maingcf = GrNewGC();
    GrSetGCForeground(state->maingcf, WHITE);
    GrSetGCBackground(state->maingcf,
                                MAIN_WINDOW_BACKGROUND_COLOUR);
    GrText(state->main_window, state->maingcf,40,35,
                "CQ SH-4/PCI nano-X Sample program ", 34, 0);


    GrMapWindow(state->main_window);
```

(a) cqsample.c

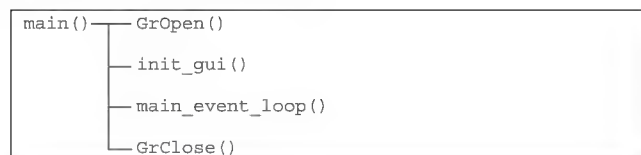
〔リスト3〕作成したサンプルプログラム(つづき)

```
    state->state = STATE_STOP;
    state->ev_cnt = 0;
    state->level = 0;
    state->running_buttons_mapped = 0;

    start_init(state);
}

void wait_for_start(cstate *state)
{
    while(state->state == STATE_STOP) {
         GrGetNextEvent(&state->event);
         handle_event(state);
    }
}

void s_start(cstate *state)
{
    while(state->state == STATE_START) {
                                        ～以下略～
```

(a) cqsample.c

```
#ifndef CQSAMPLE_H
#define CQSAMPLE_H
/*
*/
#define BORDER_WIDTH        10
#define BUTTON_HEIGHT       20
#define BUTTON_WIDTH        80
#define BUTTON_BACKGROUND_COLOUR LTGRAY
#define BUTTON_FOREGROUND_COLOUR BLACK
#define TEXT_X_POSITION     5
#define TEXT_Y_POSITION     15
#define TEXT2_Y_POSITION    30

#define MAIN_WINDOW_X_POSITION   100
#define MAIN_WINDOW_Y_POSITION   100
#define MAIN_WINDOW_WIDTH        320
#define MAIN_WINDOW_HEIGHT       240
#define MAIN_WINDOW_BACKGROUND_COLOUR CYAN

#define WINDOW_01_X_POSITION     200
#define WINDOW_01_Y_POSITION     80
#define WINDOW_01_WIDTH          100
#define WINDOW_01_HEIGHT         35
#define WINDOW_01_BACKGROUND_COLOUR BLACK
#define WINDOW_01_FOREGROUND_COLOUR GREEN

#define WINDOW_02_X_POSITION     100
#define WINDOW_02_Y_POSITION     80
#define WINDOW_02_WIDTH          100
#define WINDOW_02_HEIGHT         35
#define WINDOW_02_BACKGROUND_COLOUR WHITE
#define WINDOW_02_FOREGROUND_COLOUR GREEN

#define BUTTON_01_X_POSITION     10
#define BUTTON_01_Y_POSITION     80
#define BUTTON_01_WIDTH     BUTTON_WIDTH
#define BUTTON_01_HEIGHT    BUTTON_HEIGHT

#define BUTTON_02_X_POSITION     10
#define BUTTON_02_Y_POSITION     120
#define BUTTON_02_WIDTH     BUTTON_WIDTH
#define BUTTON_02_HEIGHT    BUTTON_HEIGHT

#define BUTTON_03_X_POSITION     100
#define BUTTON_03_Y_POSITION     180
#define BUTTON_03_WIDTH     BUTTON_WIDTH
#define BUTTON_03_HEIGHT    BUTTON_HEIGHT

#define BUTTON_04_X_POSITION     200
#define BUTTON_04_Y_POSITION     180
#define BUTTON_04_WIDTH     BUTTON_WIDTH
#define BUTTON_04_HEIGHT    BUTTON_HEIGHT

enum {
    STATE_START,
    STATE_STOP,
    STATE_EXIT,
    STATE_UNKNOWN
};

typedef GR_COLOR block;

struct cqsample_state {
    int ev_cnt;
    int ev2_cnt;
    int fhiscore;
    int level;
    int state;
    int old_state;
    int running_buttons_mapped;

    GR_WINDOW_ID main_window;
    GR_WINDOW_ID window_01;
    GR_WINDOW_ID window_02;

    GR_WINDOW_ID button_01;
    GR_WINDOW_ID button_02;
    GR_WINDOW_ID button_03;
    GR_WINDOW_ID button_04;

    GR_GC_ID maingcf;
    GR_GC_ID win_01_gcf;
    GR_GC_ID win_01_gcb;
    GR_GC_ID win_02_gcf;
    GR_GC_ID win_02_gcb;
    GR_GC_ID buttongcf;
    GR_GC_ID buttongcb;
    GR_EVENT event;
};
typedef struct cqsample_state cstate;

void draw_text(cstate *state);
void draw_01_button(cstate *state);
void draw_02_button(cstate *state);
void draw_03_button(cstate *state);
void draw_04_button(cstate *state);

void job_02_button(cstate *state);
void job_03_button(cstate *state);
void job_04_button(cstate *state);

void handle_exposure_event(cstate *state);
void handle_mouse_event(cstate *state);
void handle_keyboard_event(cstate *state);
void handle_event(cstate *state);
void start_init(cstate *state);
void init_gui(cstate *state);
void wait_for_start(cstate *state);
void s_start(cstate *state);
void main_event_loop(cstate *state);
extern size_t strlen(char *buf);

#endif
```

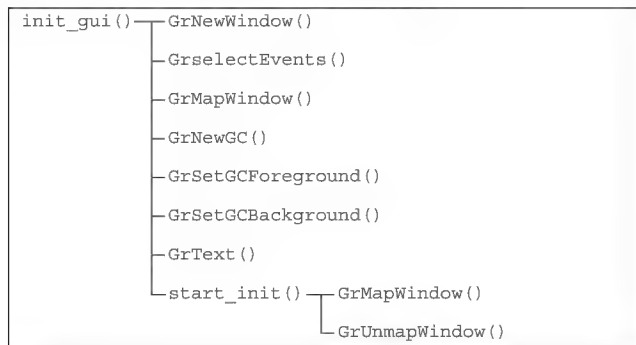
(b) cqsample.h

〔図3〕メインルーチンの構造

文献1)やリファレンスマニュアルを参照してください.

メインルーチンは**図3**のような構成になります.

▶ GrOpen()/GrClose()

nano-X(ウィンドウサーバ)に対する接続と終了です. デバイスに対するOpen()/Close()と同じで, 決まりごとと考えてく

〔図4〕ウィンドウの初期化ルーチンの構造

ださい．

▶ init_gui()——ウィンドウ初期化

ウィンドウを作成し，その中に操作ボタン，表示エリアなどの作成を行います．

▶ main_event_loop()——イベントループ

ウィンドウサーバからの情報に基づく処理を行います．処理終了後は，再びサーバからの情報を待ちます．プログラムの終了までこのプログラム内でループします．

4 ウィンドウの初期化

init_gui()関数の処理内容について説明します(**図4**)．

▶ GrNewWindow()

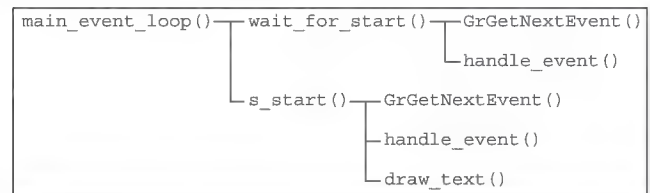
新しいウィンドウを作成します．基本は作成したいウィンドウのウィンドウIDを指定することで，そのウィンドウの中に作成されます．最初のウィンドウの場合は親ウィンドウがありません．その場合にはGR_ROOT_WINDOW_ID(実体は1)を指定します．その他の情報としては座標位置，サイズ，色などを指定します．

▶ GrselectEvents()

作成したウィンドウ(表示エリア，ボタンなど)上で発生するイベントのなかから，受け取りたいイベントを指定します．今回のプログラムで指定したイベントマスクは，次の二つです．

● GR_EVENT_MASK_EXPOSURE

再描画要求で，ウィンドウが他の操作で隠れたりした場合に発生します．必ず指定しなければいけないイベントと考えてさしつかえありません．

〔図5〕イベントループの構造

● GR_EVENT_MASK_BUTTON_DOWN

マウスボタンの操作です．

▶ GrSetGCForeground()

当該ウィンドウのフォアグランドの色指定です．

▶ GrSetGCBackground()

当該ウィンドウのバックグランドの色指定です．

▶ GrNewGC()

コンテキストIDで，画面の色，描画モードを保持し，ウィンドウIDと併用します．

▶ GrMapWindow()/GrUnmapWindow()

ウィンドウIDで指定された，ウィンドウについての表示/非表示を指定します．

▶ start_init()

プログラムの状態に対応したボタンの表示/非表示を管理します．当該処理以外でも，Start(ボタン1)/Stop(ボタン2)の扱いにより呼び出されます．

5 イベントループ

ウィンドウサーバからのイベント情報を待って，情報の提供があった場合，それに対応した処理を行います．

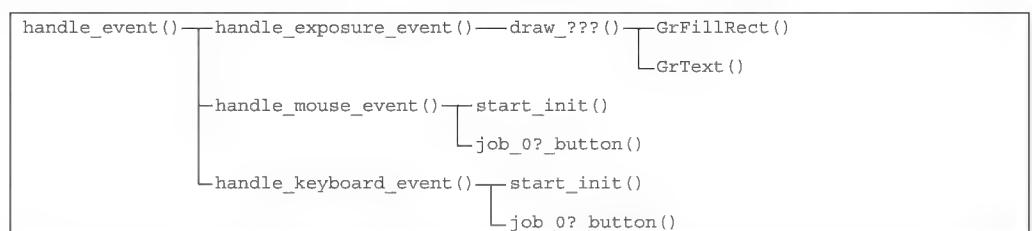
● イベントループのメイン(**図5**)

▶ wait_for_start()，s_start()

どちらもイベント待ちのループです．前者はStartボタンが押されるのを待っていて，後者はStartボタンが押された後のループです．中身はどちらもGrGetNextEvent()からのイベントを待って，イベントが届いたらhandle_event()でイベントに対応した処理を行います．

▶ GrGetNextEvent()

再描画要求，マウス，キーボードなどのイベントがなければ，内部でwaitしています．イベント発生によりGR_EVENT構造体

〔図6〕イベントハンドラの構造

〔リスト4〕起動用スクリプト

```
nano-X & sleep 1 ;  /root/cqsample
```

〔リスト5〕ターミナルウィンドウからの起動用スクリプト

```
nano-X & sleep 1 ;  nxterm & nanowm &  sleep 10000
```

(nano-X.hで定義されている)にイベント情報がセットされて復帰します.

● イベントに対応した処理(**図6**)

GrGetNextEvent()からの情報に対応した処理を行います.イベントのタイプで処理を切り分けます.

▶ handle_exposure_event()

イベントタイプがGR_EVENT_TYPE_EXPOSUREのときの処理で,exposureのwidに再描画が必要なウィンドウIDがセットされているので,そのウィンドウを再描画します.

▶ handle_mouse_event()

イベントタイプがGR_EVENT_TYPE_BUTTON_DOWNのときの処理で,mouseのwidにマウスのボタンを押したところのウィンドウのウィンドウIDがセットされています.このIDにより処理内容を決定します.

▶ handle_keyboard_event()

イベントタイプがGR_EVENT_TYPE_KEY_DOWNのときの処理で,keystrokeのchに押された文字が入力されています.今回のサンプルプログラムでは大文字の'Q'が押されたときに,プログラムを終了するようにしています.

▶ job_0?_button()

押されたボタンもしくはキーに対応した処理です.今回はfprintf()しか入っていませんが,実際のプログラムの場合は,ここにさまざまな処理を記述します.

▶ draw_???()

再描画処理です.GrFillRect()でバックグラウンドの色で塗りつぶし,その後GrText()で文字を書き込みます.

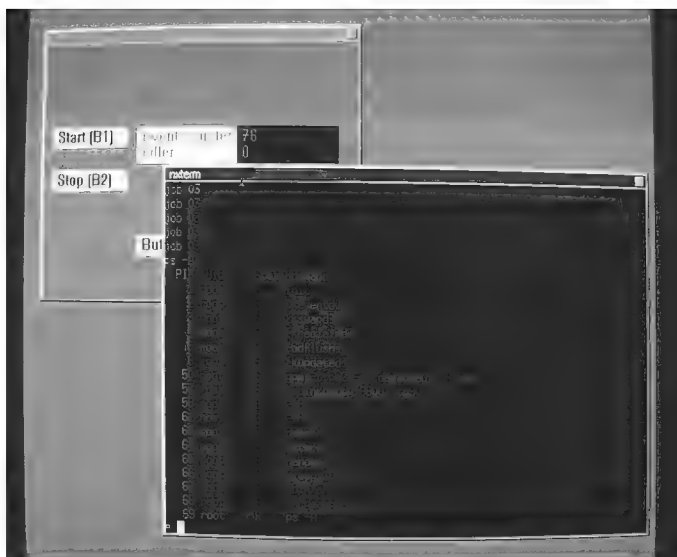
6 コンパイルと実行

● コンパイル

ソースプログラムとMakefileはmicrowindows/src/demosの下にcqsampleのディレクトリを作成し,そこに,Makefile,casample.c,cqsample.hを置きます.ここでmakeを実行するとmicrowindows/src/binにcqsampleが作成されます.

● 実行

サンプルプログラムはnano-Xのサーバが動いていることが前提になります.基本的にはnano-Xのサーバを起動し,その後,サンプルプログラムを起動するわけですが,nano-Xを起動した直後にコンソールが使えなくなります.そこで起動用そのスクリプトcqtest.sh(**リスト4**)を作成し,それを実行する形式を採ります.

〔図7〕サンプルプログラムを動作させたようす

● 実行のようす

プログラムのテストはnano-Xとサンプルプログラムだけ動作させたため,再描画のイベントが発生していないことに気付き,X Window Systemライクにstartx(**リスト5**)を作成し,ターミナルウィンドウからサンプルプログラムを起動しました.

図7にサンプルプログラムを動作させたときのようすを示します.サンプルプログラムではウィンドウの再描画ルーチンを実装していないので,ウィンドウを重ねた後に移動させると,ウィンドウが再描画されません.

今回は基本的なサンプルプログラムですが,次回の応用編では,それなりに実用に耐えるプログラム事例を解説する予定です.

まとめ

最近は,GUIのプログラムはGUIの作成ツールを使用することが一般的で,直接ウィンドウサーバのAPIを使用することはあまりないと思います.参考資料もほとんど見かけません.昔,X11でなくX10のウィンドウ時代,68020がワークステーションのメインCPUだった時代に,ワークステーションメーカーから出ていた,X Window Systemのプログラミングマニュアルがいちばんの参考書になりそうです(オークションで探すか?).

次回は,組み込み分野に応用できるように,ハードウェアと連携したGUIプログラミングについて解説します.ご期待ください.

参考文献

1) 岸 哲夫,「SHプロセッサ用Linuxの概略と組み込み用GUIの解説」,Interface増刊『Embedded UNIX』Vol.2,CQ出版(株)

さかわ・のぶひろ

音楽配信技術の最新動向

Vorbisfile APIを使用した簡単なデコーダの作成

第4回

岸 哲夫

まず，ちょっと毛色の変わった情報を紹介します．OggVorbisに対応できそうな「携帯プレーヤ」の話題です．

先ごろ，アップル社の携帯プレーヤ「iPod」にLinuxをインストールすることに成功したそうです．Linuxには，組み込み用のμClinuxを使用したようです(**図1**)．

http://www.uclinux.org/

これで，iPodでもデコーダソフトを動作させることができるはずです．iPodには浮動小数点演算ユニットがないので，整数演算のみでデコードを行う“Tremor”というライブラリを使用します．このライブラリは，公式サイトxiph.orgで配布されています．

詳細については，プロジェクト「Linux on iPod」のサイトを参照してください(**図2**)．

http://ipodlinux.sourceforge.net/

デコーダの作成

さて，前回は「Vorbisfile API」の概略を説明しました．

今回は，このVorbisfile APIを使用して簡単なデコーダを作成します．

開発環境はLinux&GNU Cです．ほかの環境でも，インクルードするヘッダを修正するだけで動作するはずです．

● ライブラリのインストール

まず，必要なライブラリをインストールします．必要なものは，下記のURLからダウンロードできます．

http://www.vorbis.com/download.psp

libao，libogg，libvorbisの三つのファイルをダウンロードしてください．

RPMも用意されていますが，tarボールをコンパイルしたほうが，自分が何をやっているかを意識できるのでよいでしょう．

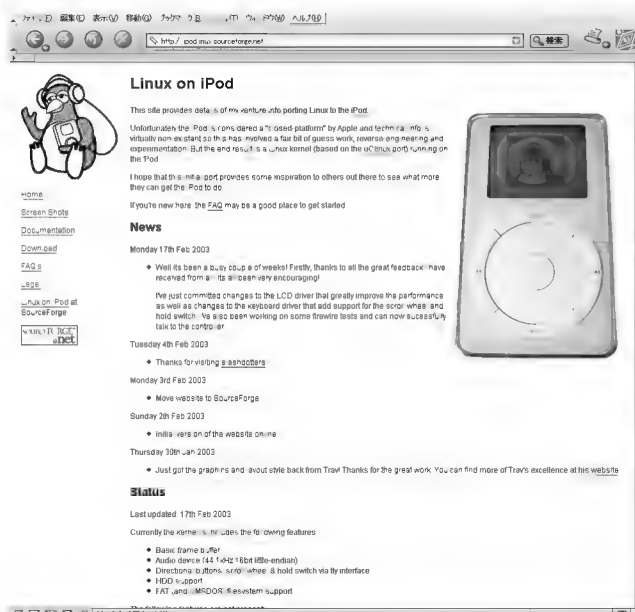
この三つのライブラリは，以下のようにインストールできます．まず，

```
tar zxvf hoge.tar.gz
```

で展開した後，できあがったディレクトリに移動します．

そこで，`./configure`を行ってください．その後，makeお

〔図1〕μClinux公式Webサイト

〔図2〕Linux on iPodのWebサイト

よびmake installで完了します．

今回のプログラムでは，OggVorbisデータをwave形式に変換するので，wave形式フォーマットの知識が必要になります．

なお，データの転送時にリトルエンディアン方式で転送しています．インテルアーキテクチャのPC以外でプログラムする際は修正してください．

● WAVE形式フォーマット（ヘッダ部）

ヘッダファイルは，**図3**のように作成してください．その後に，実際のPCMデータがつながります．

● プログラムの作成

先に述べたとおりGNU Cで作成します．プログラムを**リスト1**に示します．

ただし，このプログラムは説明のために作成したものなので，詳細なチェックを行っていません．個人の責任において使用してください．

〔図3〕WAVE形式ファイルのヘッダフォーマット

コンパイルは以下のように行います．

```
gcc -lvorbisfile  -O3 ogg2wav.c -o ogg2wav
```

もし，リンクエラーが出る場合は，ライブラリのインストールがうまく行われていないためです．再度確かめてください．

使用法は，次のとおりです．

```
ogg2wav Vorbis形式入力ファイル wave形式出力ファイル
```

● 実際に起動してみる

連載第2回で説明した，オープンオーディオライセンスに則して楽曲の配布を行っている小西健司氏のサイト(http://www.st.rim.or.jp/~ironbeat/music_wants-to_be-free/)から作品をダウンロードして，それをデコードしてみます．ダウンロードしたのは，「The_id.ogg」という曲です．

```
#ls -al *.ogg
-rwxr--r--   1 root   root   5755684
                     12月 18 23:32 The_id.ogg
#
# ./ogg2wav The_id.ogg music.wav
        [music.wav書き込みバイト数： 50872320]
```

このように，現在の書き込みバイト数を表示しながら実行します．Pentium III 800MHzの環境で，15秒で終了しました．そして結局，

```
#ls -al *.wav
-rw-r--r--   1 root   root   50872364
                      2月 24 03:35 music.wav
```

このようなwavファイルができあがります．

念のため補足すると，OggVorbisは可逆圧縮ではないので，Vorbisをwave形式に変換しても音質が元に戻るわけではありません．

では，wavファイルのヘッダ部分をダンプしてみます(**図4**)．

```
#head music.wav | od -x -w16 -A x  > dump
```

図4を見る限り，正しいWAVE形式のようです．確認のためにXMMSで聞いてみたところ，正常に再生されました．

非常に単純なプログラム例ですが，提供されたAPIを使用することで，このように簡単にデコーダが作成できます．ここに，QtなどでGUIを付けるのもよいと思います．また，このAPIは，WindowsやMacOS9，Mac OS X，BeOS，さらにはLinuxをインストールしたPlayStation2にも対応しています．

詳しくは，公式ドキュメントを参照してください．

〔図4〕wavファイルのヘッダ部分

```
000000 4952 4646 4024 0308 4157 4556 6d66 2074
000010 0010 0000 0001 0002 ac44 0000 b110 0002
000020 0004 0010 6164 6174 4000 0308 0000 0000
000030 0000 0000 0000 0000 0000 0000 0000 0000
*
006b50 0000 ffff 0000 0000 0000 0000 0000 0000
006b60 0000 0000 0000 0000 0000 0000 0000 0000
*
```

プログラムの説明

では，今回作成したogg2wav.c(**リスト1**)の内容を説明します．

● 15～20行目

インクルード文です．

20行目のヘッダファイルは，ライブラリをインストールしないとシステムに追加されないので注意してください．

● 22～26行目

プロトタイプ宣言です．なくても動きますが，付けるべきです．

● 28～35行目

データ転送用マクロです．

10進のデータを4バイト領域に16進でセットする際の変換やバイト並びの入れ替えを行っています．ビッグエンディアンの環境の場合，ここを修正します．

● 37行目

デコード時に使用するバッファの長さを設定しています．

● 42～77行目

関数write_wave_headerです．

これは，OggVorbis_Filestruct形式の構造体と出力ファイルのポインタ，全体のデータサイズを引き数にしてグローバルに置いたWAVEfmt_headbufにWAVE形式のヘッダ部を作成する関数です．130行目で行っているov_openで構造体に入力ファイルの情報が取り込まれます．チャネル数やビットレートでその情報を使用しています．

● 80～102行目

関数rewrite_wave_headerです．

ここでは，「関数write_wave_header」では計算しきれないデータの全体長を再度設定する処理をしています．

引き数は，出力ファイルのポインタと実際に読み込んだデータ長です．グローバルに置いたWAVEfmt_headbufのデータ長情報を書き換えた後，fseek(out, 0, SEEK_SET)で先頭に位置付け，再度ヘッダ部を書き換えています．

● 105～163行目

関数decode_main_procです．ここではデコードのメイン処理を行っています．

入力ファイルを開き，ov_openで初期処理をします．出力ファイルも開きます．ヘッダデータを作成し，実データをov_readで読み，変換(デコード)します．

その後，実データの長さでヘッダを再設定し，後処理をして終わります．

きし・てつお

〔**リスト1**〕**ogg2wav.c**

```
 1 //
 2 //このプログラムはOggVorbis形式のデータを
 3 //                          wav形式に変換するものです
 4 //
 5 //  ogg2wav hoge.ogg hoge.wav  とコマンドを入力してください
 6 //
 7 //
 8 //  gcc -lvorbisfile -O3 ogg2wav.c -o ogg2wav  でコンパイルできます
 9 //
10 //    なおこのプログラムは機能説明用です。
11 //           使用する場合は個人の責任において使ってください。
12 //           バグがあるかもしれません
13 //
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <getopt.h>
18 #include <errno.h>
19 #include <string.h>
20 #include <vorbis/vorbisfile.h>
21 //
22 //プロトタイプ
23 //
24 int write_wave_header(OggVorbis_File *vf, FILE *out, ogg_int64_t data_length);
25 int rewrite_wave_header(FILE *out, unsigned int write_byte);
26 static int decode_main_proc(char *infile, char *outfile);
27 //
28 //32ビット転送用マクロ
29 #define TENSOU_32(buf, x) *(buf)     = (unsigned char)((x)&0xff);¥
30                           *((buf)+1) = (unsigned char)(((x)>>8)&0xff);¥
31                           *((buf)+2) = (unsigned char)(((x)>>16)&0xff);¥
32                           *((buf)+3) = (unsigned char)(((x)>>24)&0xff);
33 //16ビット転送用マクロ
34 #define TENSOU_16(buf, x) *(buf)     = (unsigned char)((x)&0xff);¥
35                           *((buf)+1) = (unsigned char)(((x)>>8)&0xff);
36 //デコード時のバッファの長さ
37 #define  BUFLEN_   40960
38 //WAVEフォーマットのヘッダ領域
39 unsigned char WAVEfmt_headbuf[44];
40
```

〔リスト1〕**ogg2wav.c**(つづき)

```
 41 //
 42 //ここでWAVE形式のヘッダを構築します
 43 //
 44 //
 45 int write_wave_header(OggVorbis_File *vf, FILE *out, ogg_int64_t data_length)
 46 {
 47     unsigned int size = 0x7fffffff;
 48     int channels = ov_info(vf,0)->channels;
 49     int samplerate = ov_info(vf,0)->rate;
 50     int bytespersec = channels*samplerate*2;
 51     int align = channels*2;
 52     int samplesize = 16;
 53 
 54     if(data_length)         size = (unsigned int)data_length;
 55 
 56     memcpy(WAVEfmt_headbuf, "RIFF", 4);
 57     TENSOU_32(WAVEfmt_headbuf+4, size-8);
 58     memcpy(WAVEfmt_headbuf+8, "WAVE", 4);
 59     memcpy(WAVEfmt_headbuf+12, "fmt ", 4);
 60     TENSOU_32(WAVEfmt_headbuf+16, 16);
 61     TENSOU_16(WAVEfmt_headbuf+20, 1); /* format */
 62     TENSOU_16(WAVEfmt_headbuf+22, channels);
 63     TENSOU_32(WAVEfmt_headbuf+24, samplerate);
 64     TENSOU_32(WAVEfmt_headbuf+28, bytespersec);
 65     TENSOU_16(WAVEfmt_headbuf+32, align);
 66     TENSOU_16(WAVEfmt_headbuf+34, samplesize);
 67     memcpy(WAVEfmt_headbuf+36, "data", 4);
 68     TENSOU_32(WAVEfmt_headbuf+40, size - 44);
 69 
 70     if(fwrite(WAVEfmt_headbuf, 1, 44, out) != 44)
 71     {
 72         fprintf(stderr, "ヘッダファイルの書き出しに失敗しました: %s¥n", strerror(errno));
 73         return 1;
 74     }
 75 
 76     return 0;
 77 }
 78 
 79 //
 80 //WAVE形式のヘッダに処理したデータの長さを再設定します
 81 //
 82 int rewrite_wave_header(FILE *out, unsigned int write_byte)
 83 {
 84     unsigned int length = write_byte;
 85 
 86     length += 44;
 87 
 88     TENSOU_32(WAVEfmt_headbuf+4, length-8);
 89     TENSOU_32(WAVEfmt_headbuf+40, length-44);
 90     if(fseek(out, 0, SEEK_SET) != 0)
 91     {
 92         fprintf(stderr, "出力ファイルの位置付けに失敗しました: %s¥n", strerror(errno));
 93         return 1;
 94     }
 95 
 96     if(fwrite(WAVEfmt_headbuf, 1, 44, out) != 44)
 97     {
 98         fprintf(stderr, "WAV形式ファイルのヘッダーを書けませんでした: %s¥n", strerror(errno));
 99         return 1;
100     }
101     return 0;
102 }
103 
104 //
105 //デコードのメイン処理です
106 //
107 
108 static int decode_main_proc(char *infile, char *outfile)
109 {
110     FILE           *inf;
111     FILE           *outf;
112     OggVorbis_Filevf;
113     int                 bitstr = 0;
114     char           buf[BUFLEN_];
115     unsigned int  write_byte = 0;
116     int                 result;
117     ogg_int64_t         length = 0;
118     inf = fopen(infile, "rb");
119     if(!inf)
120     {
121         fprintf(stderr, "入力ファイルが開けません: %s¥n", strerror(errno));
122         return 1;
```

〔リスト1〕ogg2wav.c(つづき)

```
123     }
124     outf = fopen(outfile, "wb");
125     if(!outf) {
126         fprintf(stderr, "出力ファイルが開けません: %s¥n", strerror(errno));
127         return 1;
128     }
129
130     if(ov_open(inf, &vf, NULL, 0) < 0)
131     {
132         fprintf(stderr, "入力ファイルがVorbis形式データでは有りません¥n");
133         fclose(inf);
134         fclose(outf);
135         return 1;
136     }
137
138
139     if(write_wave_header(&vf, outf, length))
140     {
141         ov_clear(&vf);
142         fclose(outf);
143         return 1;
144     }
145
146     while((result = ov_read(&vf, buf, BUFLEN_, 0, 2, 1, &bitstr)) != 0)
147     {
148         if(fwrite(buf, 1, result, outf) != result)
149         {
150             fprintf(stderr, "出力ファイル書き込みエラー: %s¥n", strerror(errno));
151             ov_clear(&vf);
152             fclose(outf);
153             return 1;
154         }
155         write_byte += result;
156         fprintf(stderr, "¥r¥t[%s書き込みバイト数: %lld]", (char *)outfile,(long long int)write_byte);
157     }
158     fprintf(stderr, "¥n");
159     rewrite_wave_header(outf, write_byte);
160     ov_clear(&vf);
161     fclose(outf);
162     return 0;
163 }
164 //
165 //メイン処理です
166 //
167
168 int main(int argc, char **argv)
169 {
170     char infile_name[255];
171     char outfile_name[255];
172     int i;
173 //起動パラメータが1つも無い
174     if(argc == 1)
175     {
176         fprintf(stderr, "起動パラメータが一つも有りません¥n");
177         fprintf(stderr, "使用法: ogg2wav Vorbis形式入力ファイル  wave形式出力ファイル¥n");
178         return 1;
179     }
180 //起動パラメータが足りない
181     if(argc == 2)
182     {
183         fprintf(stderr, "起動パラメータが足りません¥n");
184         fprintf(stderr, "使用法: ogg2wav Vorbis形式入力ファイル  wave形式出力ファイル¥n");
185         return 1;
186     }
187     strcpy(infile_name,argv[1]);
188     strcpy(outfile_name,argv[2]);
189     if(decode_main_proc(infile_name,outfile_name))  return 1;
190     return 0;
191 }
```

第18回

ソースコードタグシステム —GNU GLOBAL

水野貴明

プログラムの書き方や命名規則などには，人それぞれに癖がある．そのため，他の人が書いたソースコードを読むのは，なかなかたいへんな作業になることが多い．また，自分が書いたソースコードであっても，しばらく経ってしまうと詳しい内容は忘れてしまい，内容を変更しようといった場合には，そのすべてを読み直す必要が出てきたりする．

そして，呼び出されている関数がどこにあるのか，変数型がどんなものなのか，その定義部分を見るためにいちいちソースコードのあちこちを探すのは，なかなかたいへんな作業である．

今回紹介するGNU GLOBALは，そんな未知の(もしくはもう忘れてしまった)ソースを読む場合に，非常に役に立つソフトウェアである．これはソースコードタグシステム，つまりソースコードを解析し，各種の定義文と，その定義が利用されている場所を調べて，相互参照を可能にするシステムである．これを利用することで，大きなソースを読む場合に，大幅な効率化が可能になる．

インストール

GNU GLOBALは，基本的にはソースコードで配布されている．GNUのダウンロードページには，各種OS向けのバイナリファイル(第三者が作成したもの)へのリンクが張られているが，今回はソースコードからインストールを行うことにした．インストールは，Linux，Windows 2000，Mac OS Xの各OSで行った．

● UNIXへのインストール

UNIX系のOSへのインストールは，他の多くのソフトウェアがそうであるように，非常に簡単である．配布されているtarballを展開したら，そのディレクトリ内に移動し，以下のように入力する．

```
./configure
make
make install
```

三つ目のmake installは，管理者権限で行う必要がある．今回は，Linux(TurboLinux 6.5 FTP版)にインストールを行ったが，何の問題もなくインストールすることができた．

● Windowsへのインストール

GNU GLOBALは，Windowsにインストールすることも可能

DATA

名称：GNU GLOBAL
作者：多摩通信社
Webサイト：http://www.gnu.org/software/global/
現在のバージョン：4.5.1
ダウンロードサイズ：440Kバイト(tarball)

である．GNU GLOBALの配布パッケージのwin32というディレクトリ内に，Windows向けにインストールを行うための方法が書かれているので，今回はそれにしたがってインストールを試みた．検証に用いたのはWindows 2000(SP3)である．

Windowsにインストールする場合も，コンパイル済みのバイナリファイルは配布されていないため，ソースからビルドする必要がある．ドキュメントは，Borland C++を利用してビルドすることを想定して書かれているので，今回はそれにしたがって，インストール作業を行うことにした．使用したのはBorland C++ v5.51である．

まず，ダウンロードしたtarballを展開し，展開されたディレクトリを「C:¥global」という名前でコピーした．続いて，以下のように入力する．

```
> C:¥global>xcopy /s w32¥*.* .
```

これは，既存のUNIX系OS用のmakeファイルなどをWindows用のもので上書きする処理なので，上書き確認が表示されるが，すべて上書きしてしまってかまわない．

コピーに続いてmakeを行う．以下のように入力する．

```
> make
```

この際，実行されるのは同じディレクトリに存在する「make.bat」というバッチファイルだが，その中から，「make.exe」というプログラムが呼び出される．このmake.exeはBorland C++とともに配布されていて，コンパイラなどと一緒に実行ファイルを入れたディレクトリ(標準では「C:¥Borland¥bcc55¥bin」)に含まれている．このディレクトリにパスが通っていれば，ビルドが正しく実行されるはずである．

ただし，Borland C++のほかに，たとえばVisualStudioなどの別の開発環境がインストールされている場合，そちらにも

〔表1〕gtagsが生成するタグファイル

ファイル名	タグファイルの中身	Apache 2.0.44でのサイズ
GTAGS	関数の定義場所	1.9Mバイト
GRTAGS	関数の参照場所	14.6Mバイト
GSYMS	その他のシンボル	5.0Mバイト
GPATH	ファイルパス	128Kバイト

〔表2〕サポートされている言語と，検索対象となる拡張子

言語名	拡張子
C	.c .h
C++	.cc .cpp .cxx .C .H
yacc	.y
Java	.java
アセンブラ	.s .S

make.exeが存在する可能性があるが，Borland C++のmake.exeでないと正しくビルドが行えない場合もある．そのような場合には，環境変数PATHに羅列されたパスのうち，前にあるものほど優先順位が高くなるので，Borland C++へのパスがもっとも優先順位が高くなるように設定するといい．

さて，コンパイル中には，警告がいくつか出てしまうが，エラーで止まることはなく，ビルドが終了する．続いてはインストール作業を行う．インストールは，以下のように入力すればいい．

```
> make install
```

ただし，標準のインストール先は「C:¥usr¥bin」(実行ファイル)と「C:¥usr¥man¥man1」(マニュアル)になっており，このディレクトリが存在していないと，インストール作業に失敗してしまう．

インストール先や，コンパイラの指定は，makefile.incに書かれているので，ここで変更が可能だが，検証ではまったく修正しなくても，正しくインストールすることができた．

続いて，その他に必要なファイルを準備する．GNU GLOBALでは，sed.exeとsort.exeという，GNUが配布しているフリーソフトウェアを必要とする．以下のサイトにそれらのソフトウェアのWindows版が公開されているので，それをダウンロードしてきた．ここで配布されているパッケージには，たくさんの実行ファイルが含まれているが，その中からsed.exeとsort.exeを選んで，「C:¥usr¥bin」にコピーした．

GNU utilities for Win32

http://www.edv.agrar.tu-muenchen.de/~syring/win32/UnxUtils.html

最後に必要なのは，Perlである．GNU GLOBALのプログラムの一部が，Perlで書かれているためだ．筆者の環境にはすでにPerl 5.6.1(ActivePerl Binary Build 633)がインストールしてあったので，新たにインストールする必要はなかった．

最後に，環境変数PATHに「C:¥usr¥bin」を追加した．この際，Windowsに標準で存在する「sort.exe(Windows 2000の場合はC:¥WINNT¥system32に置かれている)」ではなく，新しくインストールしたGNUのsort.exeがGNU GLOBALに呼び出されるようにするため，Windowsのsort.exeの置かれたディレクトリよりも「C:¥usr¥bin」を前に置く必要がある．

● Mac OS X

MacOSは，Mac OS Xになって，BSDベースのOSに生まれ変わった．そのため，UNIX用のソースをそのままコンパイルすることが可能になっている．コンパイル方法はUNIX版とまったく同じだが，標準のままインストールすると，実行ファイルが/usr/local/binに置かれるが，Mac OS Xではこのディレクトリにはパスが通っていないので，/usr/local/binにパスを通すか，以下のように，インストール先を/usr/binなどのパスが通っているディレクトリに変更する．

```
./configure --bindir=/usr/bin
```

また，Mac OS Xではsuコマンドがそのままでは利用できないので，sudoコマンドを使ってmake installを実行する．

```
sudo make install
```

なお，インストールの検証は，Mac OS X 10.2.4で行った．

データベースの生成

GNU GLOBALはいくつかのソフトウェアで構成されているが，利用するには，まずgtagsコマンドを使って，タグファイルを作成することから始める必要がある．

タグファイルを作成するには，タグを生成したいプログラムの存在するディレクトリに移動し，gtagsを実行する．

```
> gtags
```

すると，タグデータベースが生成される．生成されるファイルは**表1**の4種類である．これらのファイルは実行したディレクトリに作成される．

GNU GLOBALはC，C++，yacc[注1]，Java，アセンブラの5種類のファイルをサポートしている．gtagsが実行されると，現在のディレクトリ以下，サブディレクトリも含めて，それらのファイルを検索し，タグが作成される．ファイルタイプは拡張子を見て判断している．どの拡張子がどのファイルとみなされるかは**表2**のとおりである．

今回はサンプルとして，WebサーバApacheのバージョン2.0.44のソースを使って，タグファイルを作成してみることにした．配布されているtarballの中身すべてを対象として，タグファイルを生成した結果，対象となったのは697ファイルで，タグファイル生成にかかった時間は，筆者の環境(Pentium III-866MHz/Turbo Linux 6.5)では約1分程度だった．

各タグファイルのサイズを，**表1**に示す．Apacheのソースコードはかなり大きいため，関数の参照なども巨大なサイズになっているのがわかる．

注1：Yet Another Compiler Compiler．プログラム言語の書式を渡すと，その構文解析を行うプログラムを作成する，コンパイラを作成する際に利用するツール．

ソースコードの検索

タグファイルが生成できれば，あとはそれを利用して，定義したシンボルや関数名で検索を行うことができる．コマンドラインでの検索作業にはglobalというコマンドを利用する．

単に引き数として検索したい関数名/シンボルを指定した場合，その関数やシンボルの定義されているファイルを表示する．

```
> global main
modules/ssl/ssl_expr_scan.c
server/gen_test_char.c
(中略)
test/time-sem.c
test/zb.c
```

ファイルパスは，現在のディレクトリからの相対パスとなる．上記の例はApacheのソースコードでmain関数を検索した場合だが，ソースコード中に60個のmain関数があることがわかった．

さらに，globalにはさまざまなコマンド/オプションが用意されており，それによって表示される内容も異なる．たとえば，-xオプションでは，各シンボルの含まれる行全体とその行の行番号を表示してくれる．

また，globalではパターンの指定に，正規表現を利用することもできる．たとえば，以下のように指定すればget，もしくはsetという文字が含まれた関数名/シンボルを検索することができる．

```
> global [gs]et
```

この他にも，さまざまなオプションやコマンドが用意されており，さまざまな検索が可能になっている．たとえば，-fは特定のファイルに含まれるシンボルをすべて列挙するコマンドである．この場合，パターンではなくファイル名を指定する．

```
> global -f server/main.c
```

この他の詳しいコマンドの解説は，GNUのサイト上にあるドキュメント[注2]に書かれているので参考にするとよいだろう．

HTMLへの出力

globalコマンドを直接使うのは，特定のキーワードがすでにわかっていて，それがどこで使われているのかを調べたいときである．しかし，なにかのソースコードを初めて読むような場合，globalコマンドを使いながら，ソースコードを読み下していくのは，それはそれでなかなか大変そうである．

だが，GNU GLOBALには，そんなソースコードの読み下しを楽に行うためのツールがついている．それがhtagsである．このツールを利用すると，タグ情報とソースコードを元に，ハイパーリンクの埋め込まれたHTMLファイルとして出力することができるのだ．

htagも，やはりgtagsで作成したタグファイルを利用してHTMLを生成するので，まずはgtagsでタグファイルを作成する．その後，htagsを実行することで，そのディレクトリに「HTML」というディレクトリが作成され，その中にHTML化され，ハイパーリンクが張られたソースコードが生成されるのである．

```
> htags -s
```

何のオプションも付けなかった場合は，関数名以外のシンボルのハイパーリンクは作成されない(つまりタグファイルGSYMSは利用されない)．上記の例では，GSYMSを利用したハイパーリンクも作成するために，「-s」オプションを指定する．

作成されたディレクトリの「index.html」がスタート画面になる．**図1**ではApacheのmain関数の一覧が表示されている．

各ソースファイルのコメントには色がつけられ，他のコードとは区別されている．そして，関数名や変数型など，GNU GLOBALが検出したシンボルには，その定義場所へのリンクが張られている(複数の場所で定義されているときには，その一覧が表示される)．逆に，定義文には，そのシンボルが利用されている場所の一覧へのリンクが張られている．したがって，ソースコードを読み下していく場合に，関数の呼び出しがあれば，リンクをクリックするだけで，呼び出された関数の内容も読むことができるし，変数型も簡単に参照することができる．また，各関数の先頭部分には，そのファイルの先頭や最後，次の関数などへ移動するためのリンクが追加されている(**図2**)．

さらに，htagsには，すでに紹介した-sオプションのほかにも，いくつかのオプションがあり，必要に応じてそれらを指定することで，さまざまな表現のHTMLを作成することができる．

筆者がよく利用するのは，アルファベット順のインデックスを作成する「-a」，フレームを利用したHTMLを生成する「-F」，CGIを利用した検索機能をつけることができる「-f」である．これらをすべて設定してHTMLを生成すると，**図3**のようになる．

「-f」オプションで追加されるCGIプログラムは，global.cgiという名前で，HTMLディレクトリ内の，cgi-binディレクトリに置かれる．cgiファイルは，Perlで書かれており，実行権限がついた状態で作成されるので，Webサーバからアクセス可能なディレクトリにHTMLファイルを生成していれば，そのまま利用することができる．また，cgi-binディレクトリにはgtagsで生成したタグファイルがそのままコピーされる．このCGIプログラムは，内部的にglobalコマンドを呼び出すようになっており，これらのタグファイルはそのために利用される．

ただし，このCGIプログラムにはクロスサイトスクリプティング[注3]の問題があるので，このCGIを使ったソースコード検索機能を，インターネット上で公開する場合は，対策を講じる必

注2：http://www.gnu.org/software/global/globaldoc_toc.html
注3：CGIなどのプログラムが，渡されたデータをそのまま表示してしまうために起こる問題．もしJavaScriptなどをデータとして渡してしまうと，JavaScriptが埋め込まれたHTMLファイルがCGIによって生成されてしまう．

要があるだろう(バージョン4.51現在で確認).

なお，htags自身はPerlのスクリプトであり，内部的にglobalコマンドを発行してHTMLファイルを作成するしくみになっている．

そのほかのプログラムとの連携

GNU GLOBALの特徴は，特定のアプリケーションに依存しておらず，さまざまなアプリケーションに対応している点である．たとえば，ファイルビューアであるlessや，Emacsなどのテキストエディタ上でglobalを使うこともできる．

また，タグファイル生成時に，ファイルに保存するのではなく，フリーのデータベースであるPostgresに保存することもできるようになっているなど，それぞれの環境に合わせて，さまざまな使い方が可能になっている．

おわりに

冒頭にも述べたが，GNU GLOBALは，自分以外の誰かが書いたソースコードや，書いてから時間が経ってしまって，内容を忘れてしまったソースコードを読む際に，非常に役に立つツールである．とくにHTMLファイルを生成するhtagsは非常に便利で，GNU GLOBALの存在を知ってからというもの，筆者は何かしらのソースコードを読む際には，必ずGNU GLOBALでHTMLに加工してから読むようになったほどである．

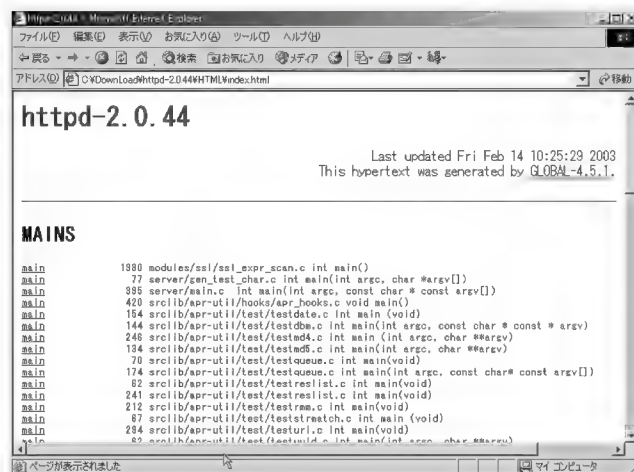
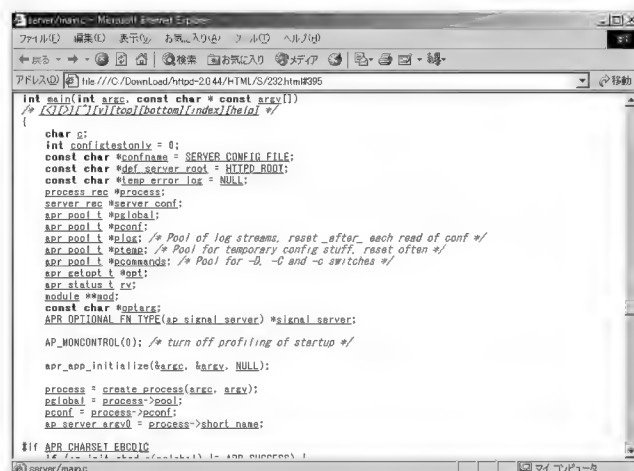
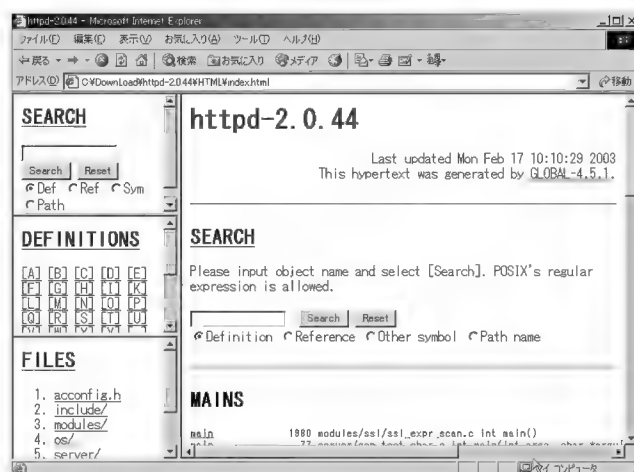
開発元である多摩通信社のページには「カーネルソースツアー」というページ[注4]があり，そこにはこのhtagsで作成したLinuxやFreeBSD，Minixなどさまざまな OSのカーネルのソースコードを見ることができる．これらのソースコードを見ると，このような大きなソースコードを読む必要があるとき，GNU GLOBALがいかに有効であるかが非常によくわかる．

「車輪の再発明」とは，すでに同じ働きをするプログラムが存在するときに，同じプログラムをいちから書き直す，といった場合などによく用いられる表現である．たしかに，現在ではさまざまなオープンソースのソフトウェアが存在し，それらのソースコードは非常に参考になるものだが，やはり他人の書いたソースを読むのがたいへんなのも事実である．しかし，GNU GLOBALを使えば，その作業が大幅に軽減されるのは間違いない．ただ，GNU GLOBALにはソースコードを整形する機能はないので，本連載で以前紹介したソースコード整形ツールであるGNU indentを併用して，ソースコード自体も自分に読みやすいよう整形することで，さらに可読性をあげることができるだろう．

こういった，ソースコードを加工するツールを有効に利用して，作業効率を高めてみてはいかがだろうか．

みずの・たかあき

注4：http://tamacom.com/tour-j.html

〔図1〕htagsが生成したHTML．トップページにはmain関数と各ファイルへのリンクが張られている

〔図2〕ソースコードには，関数，シンボルの定義場所へのリンクがつけられる

〔図3〕フレームを利用するように指定して生成したHTMLファイル．左上にはCGIによる検索機能がついている

やり直しのための

信号数学

第16回

DFTからDCTへの橋渡し

三谷政昭

これまで約2年間にわたり，DFT（ディジタルフーリエ変換と），それを高速計算するためのFFT（高速フーリエ変換）について解説してきた．

今回からは，今日の画像圧縮の国際標準規格であるJPEG（Joint Photographic Experts Group，静止画像用）やMPEG（Moving Picture Experts Group，動画像用）におけるコア技術として知られるDCT（Discrete Cosine Transform，離散コサイン変換）を採り上げることにする．

（筆者）

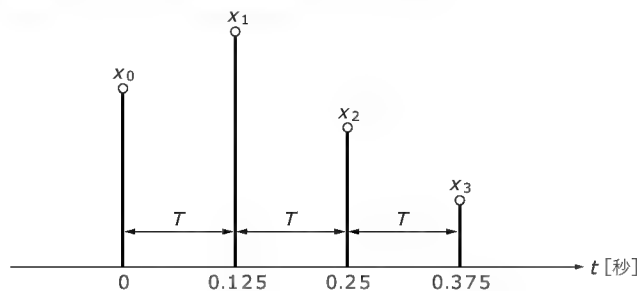
まずは，DFTからDCTを導き出すプロセスを紹介することで，DCTの基本的な考え方を理解してもらうことにする．その際，4サンプルのディジタル信号に対するDCTを例にして，数式変形の醍醐味を味わってもらうことにする．

DCT計算式のもつ意味が自然に身につくように，できるかぎりわかりやすい解説となるように配慮してあるので，数式だからといって臆することなく，じっくりと読み進めていってもらいたい（みなさんの頑張りを期待する）．

DFTと周波数の関係

最初に，簡単な例として4（＝N）サンプルのディジタル信号$\{x_0, x_1, x_2, x_3\}$に対するDFT計算式を思い起こしてもらうことから始めよう（**図16.1**）．

いま，ディジタル信号のサンプリング間隔Tが0.125［秒］とすれば，サンプリング周波数f_sは，

〔図16.1〕ディジタル信号の例（N＝4サンプル）

$$f_s = \frac{1}{T} = \frac{1}{0.125[秒]} = 8[\mathrm{Hz}] \quad \cdots\cdots (1)$$

となる．このとき，4サンプルに対する周波数スペクトル成分を表すDFT値は，

$$\begin{cases} X_0^{(4)} = \frac{1}{4}\left\{x_0 + x_1 + x_2 + x_3\right\} \\ X_1^{(4)} = \frac{1}{4}\left\{x_0 + x_1 W_4 + x_2 W_4^2 + x_3 W_4^3\right\} \\ X_2^{(4)} = \frac{1}{4}\left\{x_0 + x_1 W_4^2 + x_2 W_4^4 + x_3 W_4^6\right\} \\ X_3^{(4)} = \frac{1}{4}\left\{x_0 + x_1 W_4^3 + x_2 W_4^6 + x_3 W_4^9\right\} \end{cases} \quad \cdots\cdots (2)$$

$$ただし，\quad W_4 = e^{-j\frac{2\pi}{4}} = \cos\left(\frac{2\pi}{4}\right) - j\sin\left(\frac{2\pi}{4}\right) \quad \cdots\cdots (3)$$

と表される．ここで，DFT値を表す変数Xの上付きのカッコ内の数字はサンプル数（N＝4），下付きの数字は周波数の順番を示す．すなわち，

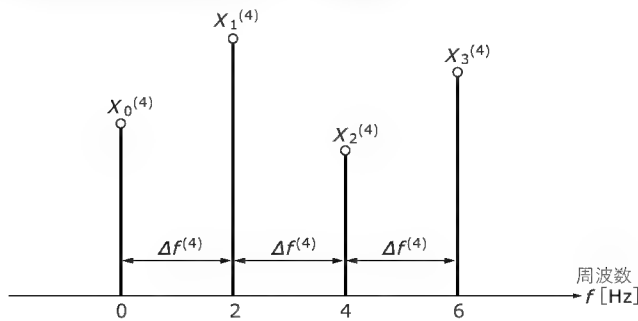
$$X_k^{(N)} \quad \cdots\cdots (4)$$

という表記は，Nサンプルのディジタル信号に対するk番目の周波数成分を意味するのである．このk番目の周波数は，サンプリング周波数f_sをN等分した周波数間隔$\Delta f^{(N)}$，すなわち，

$$\Delta f^{(N)} = \frac{f_s}{N} = \frac{1}{NT} \quad [\mathrm{Hz}] \quad \cdots\cdots (5)$$

のk倍に相当する周波数$k\Delta f^{(N)}$［Hz］を表す．つまり，$\Delta f^{(N)}$は周波数分解能に該当するわけで，**図16.1**のディジタル信号では，

$$\Delta f^{(4)} = \frac{8[\mathrm{Hz}]}{4} = 2[\mathrm{Hz}] \quad \cdots\cdots (6)$$

〔図 16.2〕周波数分解能とスペクトル成分

であり，エイリアシングを無視して考えれば，

$X_0^{(4)}$ は 0 [Hz]，すなわち直流成分
$X_1^{(4)}$ は 2 [Hz] の周波数成分
$X_2^{(4)}$ は 4 [Hz] の周波数成分
$X_3^{(4)}$ は 6 [Hz] の周波数成分

を表すことになる（**図 16.2**）．

ところで，式(2)は，$W_4^0 = 1$ を考慮して，

$$\begin{cases} X_0^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^0\right)^1 + x_2\left(W_4^0\right)^2 + x_3\left(W_4^0\right)^3\right\} \\ X_1^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^1\right)^1 + x_2\left(W_4^1\right)^2 + x_3\left(W_4^1\right)^3\right\} \\ X_2^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^2\right)^1 + x_2\left(W_4^2\right)^2 + x_3\left(W_4^2\right)^3\right\} \\ X_3^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^3\right)^1 + x_2\left(W_4^3\right)^2 + x_3\left(W_4^3\right)^3\right\} \end{cases} \quad \cdots\cdots (7)$$

のように表せることから，

$$X_k^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^k\right)^1 + x_2\left(W_4^k\right)^2 + x_3\left(W_4^k\right)^3\right\} \quad \cdots\cdots (8)$$

ただし，$k = 0, 1, 2, 3$

と一般化される．ここで，式(3)より，

$$W_4^k = \left(e^{-j\frac{2\pi}{4}}\right)^k = e^{-jk\frac{2\pi}{4}} = e^{-j2\pi\left(\frac{2k}{8}\right)} \quad \cdots\cdots (9)$$

となるので，式(1)と式(6)を考慮すれば，

$$W_4^k = e^{-j2\pi\left(\frac{k\Delta f^{(4)}}{f_s}\right)} \quad \cdots\cdots (10)$$

と表される．したがって，W_4^k は $k\Delta f^{(4)}$ の周波数を意味しているわけで，k が 0，1，2，3 の整数だけでなく，連続的な値を採るものとして考えることもできよう．なお，式(1)より，

$$\frac{1}{f_s} = T \quad \cdots\cdots (11)$$

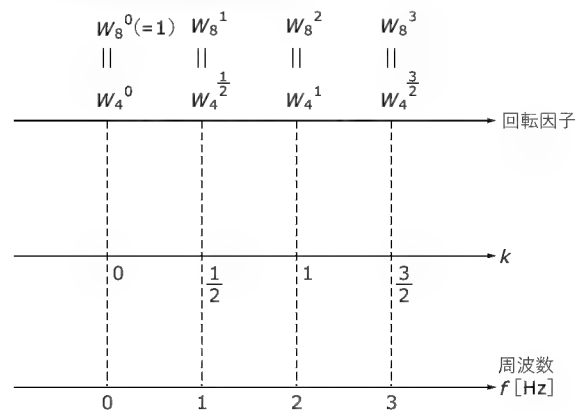
なる関係が成り立つので，式(10)は，

$$W_4^k = e^{-j2\pi\left(k\Delta f^{(4)}\right)T} \quad \cdots\cdots (12)$$

と別の表現もできる．

DFT から DCT の導出

式(12)の指数部分の $k\Delta f^{(4)}$ を連続的に変化する量として，

〔図 16.3〕周波数表現の相互関係

$$k\Delta f^{(4)} \rightarrow f[\text{Hz}] \quad \cdots\cdots (13)$$

と表せば，任意の周波数におけるスペクトル成分は，

$$\frac{1}{4}\left\{x_0 + x_1\left(e^{-j2\pi fT}\right) + x_2\left(e^{-j2\pi fT}\right)^2 + x_3\left(e^{-j2\pi fT}\right)^3\right\} \quad \cdots\cdots (14)$$

となる．

それでは，式(14)に基づき，DFT 計算式から DCT 計算式を導き出す手順を解説することにしよう．

まず，式(14)の周波数として，

$$f = 0,\ \frac{1}{2}\Delta f^{(4)},\ \Delta f^{(4)},\ \frac{3}{2}\Delta f^{(4)} \quad \cdots\cdots (15)$$

を考える．すなわち，式(10)あるいは式(12)における k の値でいえば，

$$k = 0,\ \frac{1}{2},\ 1,\ \frac{3}{2} \quad \cdots\cdots (16)$$

であり，周波数に換算すると，それぞれ，

$$f = 0,\ 1,\ 2,\ 3\ [\text{Hz}] \quad \cdots\cdots (17)$$

に相当する（**図 16.3**）．**図 16.3** より，$T = 0.125 = 1/8$ [秒] なので，

$$f = 0\,(k = 0) \rightarrow e^{-j2\pi fT} = e^{-j0} = W_4^0 = 1$$

$$f = 1\left(k = \frac{1}{2}\right) \rightarrow e^{-j2\pi fT} = e^{-j\frac{\pi}{4}} = \left(e^{-j\frac{2\pi}{4}}\right)^{\frac{1}{2}} = W_4^{\frac{1}{2}}$$

$$f = 2\,(k = 1) \rightarrow e^{-j2\pi fT} = e^{-j\frac{2\pi}{4}} = W_4^1$$

$$f = 3\left(k = \frac{3}{2}\right) \rightarrow e^{-j2\pi fT} = e^{-j\frac{3\pi}{4}} = \left(e^{-j\frac{2\pi}{4}}\right)^{\frac{3}{2}} = W_4^{\frac{3}{2}}$$

の各周波数におけるスペクトル成分を評価することを考える．

つまり，4 点の周波数（0，1，2，3 [Hz]）に対して，式(14)を書き出してみよう．以下に，得られる結果を示す．

$$\begin{cases} X_0^{(4)} = \frac{1}{4}\left\{x_0 + x_1 + x_2 + x_3\right\} \\ X_{1/2}^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^{\frac{1}{2}}\right)^1 + x_2\left(W_4^{\frac{1}{2}}\right)^2 + x_3\left(W_4^{\frac{1}{2}}\right)^3\right\} \\ X_1^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^1\right)^1 + x_2\left(W_4\right)^2 + x_3\left(W_4\right)^3\right\} \\ X_{3/2}^{(4)} = \frac{1}{4}\left\{x_0 + x_1\left(W_4^{\frac{3}{2}}\right)^1 + x_2\left(W_4^{\frac{3}{2}}\right)^2 + x_3\left(W_4^{\frac{3}{2}}\right)^3\right\} \end{cases} \quad \cdots (18)$$

ところで，

$$W_4 = e^{-j\frac{2\pi}{4}},\quad W_8 = e^{-j\frac{2\pi}{8}} \quad \cdots\cdots(19)$$

で与えられることから，

$$W_4 = e^{-j\frac{2\pi}{4}} = \left(e^{-j\frac{2\pi}{8}}\right)^2 = W_8{}^2 \quad \cdots\cdots(20)$$

となる関係が成り立つことを利用して，式(18)を書き直すことで次式が得られる．

$$\begin{cases} X_0{}^{(4)} = \frac{1}{4}\left\{x_0 + x_1 + x_2 + x_3\right\} \\ X_{1/2}{}^{(4)} = \frac{1}{4}\left\{x_0 + x_1W_8 + x_2W_8{}^2 + x_3W_8{}^3\right\} \\ X_1{}^{(4)} = \frac{1}{4}\left\{x_0 + x_1W_8{}^2 + x_2W_8{}^4 + x_3W_8{}^6\right\} \\ X_{3/2}{}^{(4)} = \frac{1}{4}\left\{x_0 + x_1W_8{}^3 + x_2W_8{}^6 + x_3W_8{}^9\right\} \end{cases} \quad \cdots\cdots(21)$$

式(21)に基づき，いよいよDCTを導き出すプロセスの説明に入ることにし，$k = 1/2$について例示する．まず，

$$\begin{aligned} X_{1/2}{}^{(4)} &= \frac{1}{8}\left\{2x_0 + 2x_1W_8 + 2x_2W_8{}^2 + 2x_3W_8{}^3\right\} \\ &= \frac{1}{8}\left\{(x_0+x_0) + (x_1+x_1)W_8 + (x_2+x_2)W_8{}^2 + (x_3+x_3)W_8{}^3\right\} \\ &= \frac{1}{8}\left\{(x_0+x_0) + (x_1+x_1)W_8 + (x_2+x_2)W_8{}^2 + (x_3+x_3)W_8{}^3 \right. \\ &\quad \left. + \underbrace{(x_3-x_3)}_{0}W_8{}^4 + \underbrace{(x_2-x_2)}_{0}W_8{}^5 + \underbrace{(x_1-x_1)}_{0}W_8{}^6 + \underbrace{(x_0-x_0)}_{0}W_8{}^7\right\} \end{aligned} \quad \cdots\cdots(22)$$

と変形する．次に，$W_8{}^{-\frac{1}{2}}$をくくり出して，

$$\begin{aligned} X_{1/2}{}^{(4)} = \frac{1}{8}W_8{}^{-\frac{1}{2}}\left\{(x_0+x_0)W_8{}^{\frac{1}{2}} + (x_1+x_1)W_8{}^{\frac{3}{2}} + (x_2+x_2)W_8{}^{\frac{5}{2}} \right. \\ + (x_3+x_3)W_8{}^{\frac{7}{2}} + (x_3-x_3)W_8{}^{\frac{9}{2}} + (x_2-x_2)W_8{}^{\frac{11}{2}} \\ \left. + (x_1-x_1)W_8{}^{\frac{13}{2}} + (x_0-x_0)W^{\frac{15}{2}}\right\} \end{aligned} \quad \cdots\cdots(23)$$

と表される．ここで，

$$W_8{}^8 = \left(e^{-j\frac{2\pi}{8}}\right)^8 = e^{-j2\pi} = \underbrace{\cos(2\pi)}_{1} - \underbrace{j\sin(2\pi)}_{0} = 1 \quad \cdots(24)$$

であることから，

$$W_8{}^{\frac{9}{2}} = W_8{}^{8-\frac{7}{2}} = W_8{}^8 \times W_8{}^{-\frac{7}{2}} = W_8{}^{-\frac{7}{2}}$$
$$W_8{}^{\frac{11}{2}} = W_8{}^{8-\frac{5}{2}} = W_8{}^8 \times W_8{}^{-\frac{5}{2}} = W_8{}^{-\frac{5}{2}}$$
$$W_8{}^{\frac{13}{2}} = W_8{}^{8-\frac{3}{2}} = W_8{}^8 \times W_8{}^{-\frac{3}{2}} = W_8{}^{-\frac{3}{2}}$$
$$W_8{}^{\frac{15}{2}} = W_8{}^{8-\frac{1}{2}} = W_8{}^8 \times W_8{}^{-\frac{1}{2}} = W_8{}^{-\frac{1}{2}}$$

となる関係が成立する．よって，式(23)は，

$$\begin{aligned} X_{1/2}{}^{(4)} = \frac{1}{8}W_8{}^{-\frac{1}{2}}\left\{(x_0+x_0)W_8{}^{\frac{1}{2}} + (x_1+x_1)W_8{}^{\frac{3}{2}} + (x_2+x_2)W_8{}^{\frac{5}{2}} \right. \\ + (x_3+x_3)W_8{}^{\frac{7}{2}} + (x_3-x_3)W_8{}^{-\frac{7}{2}} + (x_2-x_2)W_8{}^{-\frac{5}{2}} \\ \left. + (x_1-x_1)W_8{}^{-\frac{3}{2}} + (x_0-x_0)W^{-\frac{1}{2}}\right\} \end{aligned} \quad \cdots\cdots(25)$$

となり，さらに整理すると，

$$\begin{aligned} X_{1/2}{}^{(4)} = \frac{1}{8}W_8{}^{-\frac{1}{2}}\left\{x_0\left(W_8{}^{\frac{1}{2}} + W_8{}^{-\frac{1}{2}}\right) + x_1\left(W_8{}^{\frac{3}{2}} + W_8{}^{-\frac{3}{2}}\right) \right. \\ + x_2\left(W_8{}^{\frac{5}{2}} + W_8{}^{-\frac{5}{2}}\right) + x_3\left(W_8{}^{\frac{7}{2}} + W_8{}^{-\frac{7}{2}}\right) + x_0\left(W_8{}^{\frac{1}{2}} - W_8{}^{-\frac{1}{2}}\right) \\ \left. + x_1\left(W_8{}^{\frac{3}{2}} - W_8{}^{-\frac{3}{2}}\right) + x_2\left(W_8{}^{\frac{5}{2}} - W_8{}^{-\frac{5}{2}}\right) + x_3\left(W_8{}^{\frac{7}{2}} - W_8{}^{-\frac{7}{2}}\right)\right\} \end{aligned} \quad \cdots\cdots(26)$$

となる関係式が導かれる．

そこで，オイラーの公式，すなわち，

$$e^{j\theta} + e^{-j\theta} = 2\cos\theta \quad \cdots\cdots(27)$$

$$e^{j\theta} - e^{-j\theta} = j2\sin\theta \quad \cdots\cdots(28)$$

を適用すれば，たとえば，$k = 1/2$に対しては，

$$W_8{}^{\frac{1}{2}} + W_8{}^{-\frac{1}{2}} = e^{-j\frac{\pi}{8}} + e^{j\frac{\pi}{8}} = e^{j\frac{\pi}{8}} + e^{-j\frac{\pi}{8}} = 2\cos\left(\frac{\pi}{8}\right)$$

$$W_8{}^{\frac{1}{2}} - W_8{}^{-\frac{1}{2}} = e^{-j\frac{\pi}{8}} - e^{j\frac{\pi}{8}} = -\left(e^{j\frac{\pi}{8}} - e^{-j\frac{\pi}{8}}\right) = -j2\sin\left(\frac{\pi}{8}\right)$$

となる．ほかも同様にして，最終的に式(26)は，

$$\begin{aligned} X_{1/2}{}^{(4)} &= \frac{1}{8}W_8{}^{-\frac{1}{2}}\left\{2x_0\cos\left(\frac{\pi}{8}\right) + 2x_1\cos\left(\frac{3\pi}{8}\right) \right. \\ &\quad + 2x_2\cos\left(\frac{5\pi}{8}\right) + 2x_3\cos\left(\frac{7\pi}{8}\right) - j2x_0\sin\left(\frac{\pi}{8}\right) \\ &\quad \left. - j2x_1\sin\left(\frac{3\pi}{8}\right) - j2x_2\sin\left(\frac{5\pi}{8}\right) - j2x_3\sin\left(\frac{7\pi}{8}\right)\right\} \\ &= \frac{1}{4}W_8{}^{-\frac{1}{2}}\left\{x_0\cos\left(\frac{\pi}{8}\right) + x_1\cos\left(\frac{3\pi}{8}\right) + x_2\cos\left(\frac{5\pi}{8}\right) \right. \\ &\quad + x_3\cos\left(\frac{7\pi}{8}\right) - jx_0\sin\left(\frac{\pi}{8}\right) - jx_1\sin\left(\frac{3\pi}{8}\right) \\ &\quad \left. - jx_2\sin\left(\frac{5\pi}{8}\right) - jx_3\sin\left(\frac{7\pi}{8}\right)\right\} \end{aligned} \quad \cdots\cdots(29)$$

となる．さらに続けて，

$$\begin{aligned} X_{1/2}{}^{(4)} = \frac{1}{\sqrt{2}}W_8{}^{-\frac{1}{2}}\left[\frac{1}{4}\left\{\sqrt{2}x_0\cos\left(\frac{\pi}{8}\right) + \sqrt{2}x_1\cos\left(\frac{3\pi}{8}\right) \right.\right. \\ + \sqrt{2}x_2\cos\left(\frac{5\pi}{8}\right) + \sqrt{2}x_3\cos\left(\frac{7\pi}{8}\right) - j\sqrt{2}x_0\sin\left(\frac{\pi}{8}\right) \\ - j\sqrt{2}x_1\sin\left(\frac{3\pi}{8}\right) - j\sqrt{2}x_2\sin\left(\frac{5\pi}{8}\right) \\ \left.\left. - j\sqrt{2}x_3\sin\left(\frac{7\pi}{8}\right)\right\}\right] \end{aligned} \quad \cdots\cdots(30)$$

と式変形した後，

$$\begin{aligned} C_1{}^{(4)} = \frac{1}{4}\left[x_0\left\{\sqrt{2}\cos\left(\frac{\pi}{8}\right)\right\} + x_1\left\{\sqrt{2}\cos\left(\frac{3\pi}{8}\right)\right\} \right. \\ \left. + x_2\left\{\sqrt{2}\cos\left(\frac{5\pi}{8}\right)\right\} + x_3\left\{\sqrt{2}\cos\left(\frac{7\pi}{8}\right)\right\}\right] \end{aligned} \quad \cdots(31)$$

$$\begin{aligned} S_1{}^{(4)} = \frac{1}{4}\left[x_0\left\{\sqrt{2}\sin\left(\frac{\pi}{8}\right)\right\} + x_1\left\{\sqrt{2}\sin\left(\frac{3\pi}{8}\right)\right\} \right. \\ \left. + x_2\left\{\sqrt{2}\sin\left(\frac{5\pi}{8}\right)\right\} + x_3\left\{\sqrt{2}\sin\left(\frac{7\pi}{8}\right)\right\}\right] \end{aligned} \quad \cdots(32)$$

と置けば，式(30)のDFT値は，

$$X_{1/2}{}^{(4)}=\frac{1}{\sqrt{2}}W_8{}^{-\frac{1}{2}}\left[C_1{}^{(4)}-jS_1{}^{(4)}\right] \quad\cdots\cdots\cdots\cdots\cdots\cdots(33)$$

と表される．以上の変形プロセスにより，JPEGやMPEGでの画像符号化の基本的な信号変換としてのDCTが，みなさんの気づかないうちに得られているのである〔じつは式(31)がDCT〕．ついでながら，式(32)のDST(Discrete Sine Transform；離散サイン変換)も同時に導かれている．

DFTからDCT(およびDST)を導出するプロセスをフローチャートで表したものを**図16.4**に示す．

例題1

式(14)に基づき，$k=1$に対するDCT，DSTを求めよ．

解答1

図16.4のフローチャートに基づき，丁寧に計算するとよい．式(14)で，$f=2$($fT=2/8$に相当)を代入し，$W_8=e^{-j\frac{2\pi}{8}}$を用いて表す．

$$X_1{}^{(4)}=\frac{1}{8}\left\{2x_0+2x_1W_8{}^2+2x_2W_8{}^4+2x_3W_8{}^6\right\}$$

$$=\frac{1}{8}\left\{(x_0+x_0)+(x_1+x_1)W_8{}^2+(x_2+x_2)W_8{}^4+(x_3+x_3)W_8{}^6\right\}$$

$$=\frac{1}{8}\left\{(x_0+x_0)+(x_1+x_1)W_8{}^2+(x_2+x_2)W_8{}^4+(x_3+x_3)W_8{}^6+\underbrace{(x_3-x_3)}_{0}W_8{}^8+\underbrace{(x_2-x_2)}_{0}W_8{}^{10}+\underbrace{(x_1-x_1)}_{0}W_8{}^{12}+\underbrace{(x_0-x_0)}_{0}W_8{}^{14}\right\} \quad\cdots\cdots\cdots(34)$$

次に，$W_8{}^{-1}$をくくり出す．

$$X_1{}^{(4)}=\frac{1}{8}W_8{}^{-1}\left\{(x_0+x_0)W_8{}^1+(x_1+x_1)W_8{}^3+(x_2+x_2)W_8{}^5+(x_3+x_3)W_8{}^7+(x_3-x_3)W_8{}^9+(x_2-x_2)W_8{}^{11}+(x_1-x_1)W_8{}^{13}+(x_0-x_0)W^{15}\right\}$$

$$=\frac{1}{8}W_8{}^{-1}\left\{(x_0+x_0)W_8{}^1+(x_1+x_1)W_8{}^3+(x_2+x_2)W_8{}^5+(x_3+x_3)W_8{}^7+(x_3-x_3)W_8{}^{-7}+(x_2-x_2)W_8{}^{-5}+(x_1-x_1)W_8{}^{-3}+(x_0-x_0)W^{-1}\right\}$$

$$=\frac{1}{8}W_8{}^{-1}\left\{x_0\left(W_8{}^1+W_8{}^{-1}\right)+x_1\left(W_8{}^3+W_8{}^{-3}\right)+x_2\left(W_8{}^5+W_8{}^{-5}\right)+x_3\left(W_8{}^7+W_8{}^{-7}\right)+x_0\left(W_8{}^1-W_8{}^{-1}\right)+x_1\left(W_8{}^3-W_8{}^{-3}\right)+x_2\left(W_8{}^5-W_8{}^{-5}\right)+x_3\left(W_8{}^7-W_8{}^{-7}\right)\right\}$$

$$=\frac{1}{8}W_8{}^{-1}\left\{2x_0\cos\left(\frac{2\pi}{8}\right)+2x_1\cos\left(\frac{6\pi}{8}\right)+2x_2\cos\left(\frac{10\pi}{8}\right)+2x_3\cos\left(\frac{14\pi}{8}\right)-j2x_0\sin\left(\frac{2\pi}{8}\right)-j2x_1\sin\left(\frac{6\pi}{8}\right)-j2x_2\sin\left(\frac{10\pi}{8}\right)-j2x_3\sin\left(\frac{14\pi}{8}\right)\right\}$$

〔**図16.4**〕**DFTからDCT，DSTを導出するプロセス**($N=4$，$\ell\neq0$)

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{4}\left[x_0+x_1\left(W_4{}^{\frac{\ell}{2}}\right)+x_2\left(W_4{}^{\frac{\ell}{2}}\right)^2+x_3\left(W_4{}^{\frac{\ell}{2}}\right)^3\right]$$

⇩ $W_4=W_8{}^2$を代入

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{4}\left[x_0+x_1\left(W_8{}^{\ell}\right)+x_2\left(W_8{}^{\ell}\right)^2+x_3\left(W_8{}^{\ell}\right)^3\right]$$

⇩

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}\left[2x_0+2x_1W_8{}^{\ell}+2x_2W_8{}^{2\ell}+2x_3W_8{}^{3\ell}\right]$$

⇩ 対称化，反対称化

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}\left[(x_0+x_0)+(x_1+x_1)W_8{}^{\ell}+(x_2+x_2)W_8{}^{2\ell}+(x_3+x_3)W_8{}^{3\ell}+(x_3-x_3)W_8{}^{4\ell}+(x_2-x_2)W_8{}^{5\ell}+(x_1-x_1)W_8{}^{6\ell}+(x_0-x_0)W_8{}^{7\ell}\right]$$

⇩ $W_8{}^{-\frac{\ell}{2}}$をくくりだす

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}W_8{}^{-\frac{\ell}{2}}\left[(x_0+x_0)W_8{}^{\frac{\ell}{2}}+(x_1+x_1)W_8{}^{\frac{3\ell}{2}}+(x_2+x_2)W_8{}^{\frac{5\ell}{2}}+(x_3+x_3)W_8{}^{\frac{7\ell}{2}}+(x_3-x_3)W_8{}^{\frac{9\ell}{2}}+(x_2-x_2)W_8{}^{\frac{11\ell}{2}}+(x_1-x_1)W_8{}^{\frac{13\ell}{2}}+(x_0-x_0)W_8{}^{\frac{15\ell}{2}}\right]$$

⇩

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}W_8{}^{-\frac{\ell}{2}}\left[x_0(W_8{}^{\frac{\ell}{2}}+W_8{}^{\frac{15\ell}{2}})+x_1(W_8{}^{\frac{3\ell}{2}}+W_8{}^{\frac{13\ell}{2}})+x_2(W_8{}^{\frac{5\ell}{2}}+W_8{}^{\frac{11\ell}{2}})+x_3(W_8{}^{\frac{7\ell}{2}}+W_8{}^{\frac{9\ell}{2}})+x_0(W_8{}^{\frac{\ell}{2}}-W_8{}^{\frac{15\ell}{2}})+x_1(W_8{}^{\frac{3\ell}{2}}-W_8{}^{\frac{13\ell}{2}})+x_2(W_8{}^{\frac{5\ell}{2}}-W_8{}^{\frac{11\ell}{2}})+x_3(W_8{}^{\frac{7\ell}{2}}-W_8{}^{\frac{9\ell}{2}})\right]$$

⇩ $W_8{}^{\frac{m\ell}{2}}=W_8{}^{\left(8-\frac{m}{2}\right)\ell}=W_8{}^{-\frac{m\ell}{2}}$

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}\left[x_0(W_8{}^{\frac{\ell}{2}}+W_8{}^{-\frac{\ell}{2}})+x_1(W_8{}^{\frac{3\ell}{2}}+W_8{}^{-\frac{3\ell}{2}})+x_2(W_8{}^{\frac{5\ell}{2}}+W_8{}^{-\frac{5\ell}{2}})+x_3(W_8{}^{\frac{7\ell}{2}}+W_8{}^{-\frac{7\ell}{2}})+x_0(W_8{}^{\frac{\ell}{2}}-W_8{}^{-\frac{\ell}{2}})+x_1(W_8{}^{\frac{3\ell}{2}}-W_8{}^{-\frac{3\ell}{2}})+x_2(W_8{}^{\frac{5\ell}{2}}-W_8{}^{-\frac{5\ell}{2}})+x_3(W_8{}^{\frac{7\ell}{2}}-W_8{}^{-\frac{7\ell}{2}})\right]$$

⇩ オイラーの公式[式(27)，式(28)]による

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{8}\left[2x_0\cos\left(\frac{\ell\pi}{8}\right)+2x_1\cos\left(\frac{3\ell\pi}{8}\right)+2x_2\cos\left(\frac{5\ell\pi}{8}\right)+2x_3\cos\left(\frac{7\ell\pi}{8}\right)-j\,2x_0\sin\left(\frac{\ell\pi}{8}\right)-j\,2x_1\sin\left(\frac{3\ell\pi}{8}\right)-j\,2x_2\sin\left(\frac{5\ell\pi}{8}\right)-j\,2x_3\sin\left(\frac{7\ell\pi}{8}\right)\right]$$

⇩

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{4}\left[\left\{x_0\cos\left(\frac{\ell\pi}{8}\right)+x_1\cos\left(\frac{3\ell\pi}{8}\right)+x_2\cos\left(\frac{5\ell\pi}{8}\right)+x_3\cos\left(\frac{7\ell\pi}{8}\right)\right\}-j\left\{x_0\sin\left(\frac{\ell\pi}{8}\right)+x_1\sin\left(\frac{3\ell\pi}{8}\right)+x_2\sin\left(\frac{5\ell\pi}{8}\right)+x_2\sin\left(\frac{7\ell\pi}{8}\right)\right\}\right]$$

⇩ 正規化

$$X_{\frac{\ell}{2}}{}^{(4)}=\frac{1}{\sqrt{2}}\left[\frac{1}{4}\left\{\sqrt{2}\,x_0\cos\left(\frac{\ell\pi}{8}\right)+\sqrt{2}\,x_1\cos\left(\frac{3\ell\pi}{8}\right)+\sqrt{2}\,x_2\cos\left(\frac{5\ell\pi}{8}\right)+\sqrt{2}\,x_3\cos\left(\frac{7\ell\pi}{8}\right)\right\}-j\frac{1}{4}\left\{\sqrt{2}\,x_0\sin\left(\frac{\ell\pi}{8}\right)+\sqrt{2}\,x_1\sin\left(\frac{3\ell\pi}{8}\right)+\sqrt{2}\,x_2\sin\left(\frac{5\ell\pi}{8}\right)+\sqrt{2}\,x_3\sin\left(\frac{7\ell\pi}{8}\right)\right\}\right]$$

⇩

$$\text{DCT}:C_{\ell}{}^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\cos\left(\frac{\ell\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\cos\left(\frac{3\ell\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\cos\left(\frac{5\ell\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\cos\left(\frac{7\ell\pi}{8}\right)\right\}\right]$$

$$\text{DST}:S_{\ell}{}^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\sin\left(\frac{\ell\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\sin\left(\frac{3\ell\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\sin\left(\frac{5\ell\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\sin\left(\frac{7\ell\pi}{8}\right)\right\}\right]$$

$$=\frac{1}{4}W_8^{-1}\left\{x_0\cos\left(\frac{2\pi}{8}\right)+x_1\cos\left(\frac{6\pi}{8}\right)+x_2\cos\left(\frac{10\pi}{8}\right)+x_3\cos\left(\frac{14\pi}{8}\right)-jx_0\sin\left(\frac{2\pi}{8}\right)-jx_1\sin\left(\frac{6\pi}{8}\right)-jx_2\sin\left(\frac{10\pi}{8}\right)-jx_3\sin\left(\frac{14\pi}{8}\right)\right\} \cdots\cdots(35)$$

最終的に，次式が得られる．

$$X_1^{(4)}=\frac{1}{\sqrt{2}}W_8^{-1}\left[\frac{1}{4}\left\{\sqrt{2}x_0\cos\left(\frac{2\pi}{8}\right)+\sqrt{2}x_1\cos\left(\frac{6\pi}{8}\right)+\sqrt{2}x_2\cos\left(\frac{10\pi}{8}\right)+\sqrt{2}x_3\cos\left(\frac{14\pi}{8}\right)-j\sqrt{2}x_0\sin\left(\frac{2\pi}{8}\right)-j\sqrt{2}x_1\sin\left(\frac{6\pi}{8}\right)-j\sqrt{2}x_2\sin\left(\frac{10\pi}{8}\right)-j\sqrt{2}x_3\sin\left(\frac{14\pi}{8}\right)\right\}\right] \cdots(36)$$

よって，DCTとDSTはそれぞれ以下のようになる．

$$X_1^{(4)}=\frac{1}{\sqrt{2}}W_8^{-1}\left[C_2^{(1)}-jS_2^{(4)}\right] \cdots\cdots(37)$$

ただし，

$$C_2^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\cos\left(\frac{2\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\cos\left(\frac{6\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\cos\left(\frac{10\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\cos\left(\frac{14\pi}{8}\right)\right\}\right] \cdots(38)$$

$$S_2^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\sin\left(\frac{2\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\sin\left(\frac{6\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\sin\left(\frac{10\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\sin\left(\frac{14\pi}{8}\right)\right\}\right] \cdots(39)$$

なお，$k=0$，$3/2$についても結果のみを示しておくので，各自で必ず検証しておいてもらいたい．

(i) $k=0$(直流)の場合

$$X_0^{(4)}=C_0^{(4)}-jS_0^{(4)} \cdots\cdots(40)$$

ただし，

$$C_0^{(4)}=\frac{1}{4}\left[x_0\{1\}+x_1\{1\}+x_2\{1\}+x_3\{1\}\right] \cdots\cdots(41)$$

$$S_0^{(4)}=0 \cdots\cdots(42)$$

(ii) $k=3/2$の場合

$$X_{3/2}^{(4)}=\frac{1}{\sqrt{2}}W_8^{-\frac{3}{2}}\left[C_3^{(4)}-jS_3^{(4)}\right] \cdots\cdots(43)$$

ただし，

$$C_3^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\cos\left(\frac{3\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\cos\left(\frac{9\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\cos\left(\frac{15\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\cos\left(\frac{21\pi}{8}\right)\right\}\right] \cdots(44)$$

$$S_3^{(4)}=\frac{1}{4}\left[x_0\left\{\sqrt{2}\sin\left(\frac{3\pi}{8}\right)\right\}+x_1\left\{\sqrt{2}\sin\left(\frac{9\pi}{8}\right)\right\}+x_2\left\{\sqrt{2}\sin\left(\frac{15\pi}{8}\right)\right\}+x_3\left\{\sqrt{2}\sin\left(\frac{21\pi}{8}\right)\right\}\right] \cdots(45)$$

DCTの正規直交基底ベクトル

まず，式(31)，式(38)，式(41)，式(44)における{ }の中の4個の値を基底ベクトルの要素として，

$$\boldsymbol{\phi}^{(0)}=\{1,\ 1,\ 1,\ 1\} \cdots\cdots(46)$$

$$\boldsymbol{\phi}^{(1)}=\left\{\sqrt{2}\cos\left(\frac{\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{3\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{5\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{7\pi}{8}\right)\right\} \cdots\cdots(47)$$

$$\boldsymbol{\phi}^{(2)}=\left\{\sqrt{2}\cos\left(\frac{2\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{6\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{10\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{14\pi}{8}\right)\right\} \cdots\cdots(48)$$

$$\boldsymbol{\phi}^{(3)}=\left\{\sqrt{2}\cos\left(\frac{3\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{9\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{15\pi}{8}\right),\ \sqrt{2}\cos\left(\frac{21\pi}{8}\right)\right\} \cdots\cdots(49)$$

と定義する．手始めに，式(46)～式(49)の各基底ベクトルのノルム$\|\boldsymbol{\phi}^{(k)}\|$を計算してみよう(2001年7月号，第2回「正規直交基底とディジタル信号解析」参照)．ここでは，4次元ベクトル$\boldsymbol{p}=\{p_1,\ p_2,\ p_3,\ p_4\}$のノルム$\|\boldsymbol{p}\|$を1サンプルあたりとして，

$$\|\boldsymbol{p}\|=\sqrt{\frac{1}{4}\left\{(p_1)^2+(p_2)^2+(p_3)^2+(p_4)^2\right\}}$$

で定義している．よって，各基底ベクトルのノルムはそれぞれ，

$$\|\boldsymbol{\phi}^{(0)}\|=\sqrt{\frac{1}{4}\left(1^2+1^2+1^2+1^2\right)}=1 \cdots\cdots(50)$$

$$\|\boldsymbol{\phi}^{(1)}\|=\sqrt{\frac{1}{4}\left[\left\{\sqrt{2}\cos\left(\frac{\pi}{8}\right)\right\}^2+\left\{\sqrt{2}\cos\left(\frac{3\pi}{8}\right)\right\}^2+\left\{\sqrt{2}\cos\left(\frac{5\pi}{8}\right)\right\}^2+\left\{\sqrt{2}\cos\left(\frac{7\pi}{8}\right)\right\}^2\right]}$$
$$=\frac{1}{\sqrt{2}}\sqrt{\cos^2\left(\frac{\pi}{8}\right)+\cos^2\left(\frac{3\pi}{8}\right)+\cos^2\left(\frac{5\pi}{8}\right)+\cos^2\left(\frac{7\pi}{8}\right)} \cdots(51)$$

$$\|\boldsymbol{\phi}^{(2)}\|=\frac{1}{\sqrt{2}}\sqrt{\cos^2\left(\frac{2\pi}{8}\right)+\cos^2\left(\frac{6\pi}{8}\right)+\cos^2\left(\frac{10\pi}{8}\right)+\cos^2\left(\frac{14\pi}{8}\right)} \cdots\cdots(52)$$

$$\|\boldsymbol{\phi}^{(3)}\|=\frac{1}{\sqrt{2}}\sqrt{\cos^2\left(\frac{3\pi}{8}\right)+\cos^2\left(\frac{9\pi}{8}\right)+\cos^2\left(\frac{15\pi}{8}\right)+\cos^2\left(\frac{21\pi}{8}\right)} \cdots\cdots(53)$$

となる．式(51)～式(53)については，

$$\frac{\pi}{8}(k{=}1),\ \frac{2\pi}{8}(k{=}2),\ \frac{3\pi}{8}(k{=}3) \quad\cdots\cdots(54)$$

の三つの角度を仮にαと置くことにすれば，いずれのノルムも，

$$\left\|\boldsymbol{\phi}^{(k)}\right\| = \frac{1}{\sqrt{2}}\sqrt{\cos^2(\alpha)+\cos^2(3\alpha)+\cos^2(5\alpha)+\cos^2(7\alpha)} \quad\cdots\cdots(55)$$

の形式で表される．このとき，式(27)の関係から，

$$\begin{cases} \cos^2(\alpha)=\left(\dfrac{e^{j\alpha}+e^{-j\alpha}}{2}\right)^2=\dfrac{e^{j2\alpha}+2+e^{-j2\alpha}}{4} \\ \cos^2(3\alpha)=\left(\dfrac{e^{j3\alpha}+e^{-j3\alpha}}{2}\right)^2=\dfrac{e^{j6\alpha}+2+e^{-j6\alpha}}{4} \\ \cos^2(5\alpha)=\left(\dfrac{e^{j5\alpha}+e^{-j5\alpha}}{2}\right)^2=\dfrac{e^{j10\alpha}+2+e^{-j10\alpha}}{4} \\ \cos^2(7\alpha)=\left(\dfrac{e^{j7\alpha}+e^{-j7\alpha}}{2}\right)^2=\dfrac{e^{j14\alpha}+2+e^{-j14\alpha}}{4} \end{cases} \quad\cdots(56)$$

の複素数による表現を用いることにより，根号(√)の中は，

$$\cos^2(\alpha)+\cos^2(3\alpha)+\cos^2(5\alpha)+\cos^2(7\alpha)$$
$$=\frac{8+e^{j2\alpha}\left\{1+e^{j4\alpha}+\left(e^{j4\alpha}\right)^2+\left(e^{j4\alpha}\right)^3\right\}}{4}$$
$$+\frac{e^{-j2\alpha}\left\{1+e^{-j4\alpha}+\left(e^{-j4\alpha}\right)^2+\left(e^{-j4\alpha}\right)^3\right\}}{4} \quad\cdots\cdots(57)$$

となる．

また，式(57)の{ }の中は等比級数なので，

$$1+e^{j4\alpha}+\left(e^{j4\alpha}\right)^2+\left(e^{j4\alpha}\right)^3=\frac{1-\left(e^{j4\alpha}\right)^4}{1-e^{j4\alpha}}=\frac{1-e^{j16\alpha}}{1-e^{j4\alpha}} \quad\cdots\cdots(58)$$

$$1+e^{-j4\alpha}+\left(e^{-j4\alpha}\right)^2+\left(e^{-j4\alpha}\right)^3=\frac{1-\left(e^{-j4\alpha}\right)^4}{1-e^{-j4\alpha}}=\frac{1-e^{-j16\alpha}}{1-e^{-j4\alpha}} \quad\cdots\cdots(59)$$

と計算される．このとき，式(54)の値に対してはすべて，式(58)，式(59)において，式(24)より，

$$e^{j16\alpha}=(e^{j2\pi})^k=1^k=1$$
$$e^{-j16\alpha}=(e^{-j2\pi})^k=1^k=1$$

であることから，式(57)の{ }の中はいずれも0になる．以上の計算より，式(55)は，$k=1,\ 2,\ 3$に対していずれも，

$$\left\|\boldsymbol{\phi}^{(k)}\right\| = \frac{1}{\sqrt{2}}\sqrt{\frac{8+0+0}{4}}=1 \quad\cdots\cdots(60)$$

とノルムが1であることになり，「正規化」されていることがわかる．

また，詳細な計算は省略するが，相異なる基底ベクトルの内積，すなわち，

$$\begin{cases} \left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(1)}\right\rangle=\left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(2)}\right\rangle=\left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(3)}\right\rangle=0 \\ \left\langle\boldsymbol{\phi}^{(1)},\boldsymbol{\phi}^{(2)}\right\rangle=\left\langle\boldsymbol{\phi}^{(1)},\boldsymbol{\phi}^{(3)}\right\rangle=\left\langle\boldsymbol{\phi}^{(2)},\boldsymbol{\phi}^{(3)}\right\rangle=0 \end{cases} \quad\cdots\cdots(61)$$

がすべて‘0’であることから，「直交性」の条件を満たすことも容易に確かめられる．

よって，4個の基底ベクトル$\{\boldsymbol{\phi}^{(0)},\ \boldsymbol{\phi}^{(1)},\ \boldsymbol{\phi}^{(2)},\ \boldsymbol{\phi}^{(3)}\}$は，$\ell$，$m=0,\ 1,\ 2,\ 3$に対して，

$$\left\langle\boldsymbol{\phi}^{(\ell)},\boldsymbol{\phi}^{(m)}\right\rangle=\begin{cases}1 & ;\ell=m \\ 0 & ;\ell\neq m\end{cases} \quad\cdots\cdots(62)$$

であり，「正規性」と「直交性」の二つの条件を同時に満足しているわけで，「正規直交基底」となるのである．

例題2

式(46)～式(49)の基底ベクトルの「直交性」，すなわち式(61)の関係が成立することを検証せよ．

解答2

まず，二つの4次元ベクトル$\boldsymbol{p}=\{p_1,\ p_2,\ p_3,\ p_4\}$と$\boldsymbol{q}=\{q_1,\ q_2,\ q_3,\ q_4\}$の内積は，

$$\langle\boldsymbol{p},\boldsymbol{q}\rangle=\frac{1}{4}\left(p_1q_1+p_2q_2+p_3q_3+p_4q_4\right) \quad\cdots\cdots(63)$$

で定義される．たとえば，$\ell=0$，$m=1$のときは，式(63)より，式(46)と式(47)を適用して，

$$\left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(1)}\right\rangle=$$
$$\frac{1}{4}\left\{1\times\sqrt{2}\cos\left(\frac{\pi}{8}\right)+1\times\sqrt{2}\cos\left(\frac{3\pi}{8}\right)\right.$$
$$\left.+1\times\sqrt{2}\cos\left(\frac{5\pi}{8}\right)+1\times\sqrt{2}\cos\left(\frac{7\pi}{8}\right)\right\}$$
$$=\frac{\sqrt{2}}{4}\left\{\cos\left(\frac{\pi}{8}\right)+\cos\left(\frac{3\pi}{8}\right)+\cos\left(\frac{5\pi}{8}\right)+\cos\left(\frac{7\pi}{8}\right)\right\} \quad\cdots(64)$$

と表される．ここで，式(64)の{ }の中は，

$$\frac{\pi}{8}=\alpha \quad\cdots\cdots(65)$$

と置くことにより，

$$\cos(\alpha)+\cos(3\alpha)+\cos(5\alpha)+\cos(7\alpha) \quad\cdots\cdots(66)$$

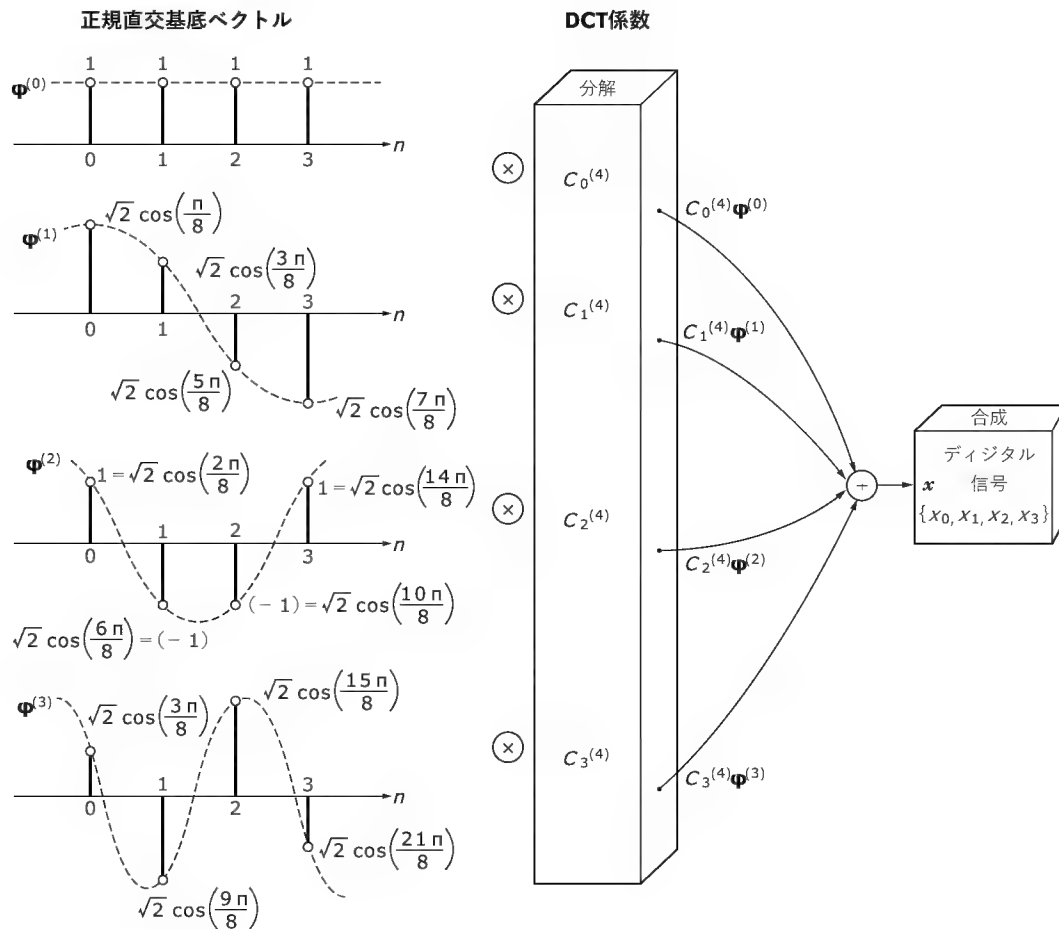
と表される．次に，式(27)の関係から，式(66)は，

$$\frac{e^{j\alpha}+e^{-j\alpha}}{2}+\frac{e^{j3\alpha}+e^{-j3\alpha}}{2}+\frac{e^{j5\alpha}+e^{-j5\alpha}}{2}+\frac{e^{j7\alpha}+e^{-j7\alpha}}{2}$$
$$=\frac{e^{j\alpha}\left\{1+e^{j2\alpha}+\left(e^{j2\alpha}\right)^2+\left(e^{j2\alpha}\right)^3\right\}}{2}$$
$$+\frac{e^{-j\alpha}\left\{1+e^{-j\alpha}+\left(e^{-j2\alpha}\right)^2+\left(e^{-j2\alpha}\right)^3\right\}}{2} \quad\cdots\cdots(67)$$

となる．さらに続けて，式(67)の{ }の中は等比級数なので，式(58)，式(59)と同様の計算により，

$$1+e^{j2\alpha}+\left(e^{j2\alpha}\right)^2+\left(e^{j2\alpha}\right)^3=\frac{1-\left(e^{j2\alpha}\right)^4}{1-e^{j2\alpha}}=\frac{1-e^{j8\alpha}}{1-e^{j2\alpha}} \quad\cdots\cdots(68)$$

$$1+e^{-j2\alpha}+\left(e^{-j2\alpha}\right)^2+\left(e^{-j2\alpha}\right)^3=\frac{1-\left(e^{-j2\alpha}\right)^4}{1-e^{-j2\alpha}}=\frac{1-e^{-j8\alpha}}{1-e^{-j2\alpha}} \quad\cdots\cdots(69)$$

〔図16.5〕ディジタル信号のDCT分解/合成

となり，式(67)の分子は，

$$e^{j\alpha}\frac{1-e^{j8\alpha}}{1-e^{j2\alpha}}+e^{-j\alpha}\frac{1-e^{-j8\alpha}}{1-e^{-j2\alpha}}$$
$$=\frac{1-e^{j8\alpha}}{e^{-j\alpha}-e^{j\alpha}}+\frac{1-e^{-j8\alpha}}{e^{j\alpha}-e^{-j\alpha}}$$
$$=-\frac{1-e^{j8\alpha}}{e^{j\alpha}-e^{-j\alpha}}+\frac{1-e^{-j8\alpha}}{e^{j\alpha}-e^{-j\alpha}}$$
$$=\frac{e^{j8\alpha}-e^{-j8\alpha}}{e^{j\alpha}-e^{-j\alpha}} \quad \cdots\cdots\cdots\cdots(70)$$

と変形できる．ここで，式(65)より，

$$e^{j8\alpha}=e^{j\pi}=-1$$
$$e^{-j8\alpha}=e^{-j\pi}=-1$$

であることから，式(70)は‘0’に等しく，

$$\phi\langle {}^{(0)},\ \phi^{(1)}\rangle=0$$

となるので，$\phi^{(0)}$と$\phi^{(1)}$とは「直交」することが確かめられる．そのほかの場合もすべて‘0’となることは，各自で計算して確かめられたい．

DCT値は正規直交基底ベクトルによる展開係数

ここでは，DCT値$\{C_k^{(4)}\}_{k=0}^{k=3}$が正規直交基底ベクトルによる展開係数であることを説明する．

たとえば，式(31)のDCT値$C_1^{(4)}$を考えてみよう．信号ベクトルを$x=\{x_0,\ x_1,\ x_2,\ x_3\}$と表せば，式(63)の内積の定義より，式(47)の正規直交基底ベクトル$\phi^{(1)}$と信号ベクトルxとの内積が$C_1^{(4)}$に一致することは容易におわかりなるであろう．

同様に考えて，ディジタル信号$\{x_0,\ x_1,\ x_2,\ x_3\}$に対するDCT値，すなわち式(31)，式(38)，式(41)，式(44)は，式(46)～式(49)の正規直交基底ベクトル$\{\phi^{(0)},\ \phi^{(1)},\ \phi^{(2)},\ \phi^{(3)}\}$を用いて，以下のように内積として記述される．

$$\begin{cases} C_0^{(4)}=\langle x,\ \phi^{(0)}\rangle & C_1^{(4)}=\langle \phi \quad {}^{(1)}\rangle \\ C_2^{(4)}=\langle x,\ \phi^{(2)}\rangle & C_3^{(4)}=\langle \phi \quad {}^{(3)}\rangle \end{cases} \quad \cdots\cdots\cdots\cdots(71)$$

言い換えれば，式(71)で与えられる内積がDCT値そのもので

あり，正規直交基底ベクトルによる展開係数に一致しており，信号ベクトル $\boldsymbol{x}$ は正規直交基底ベクトルの線形結合(一次結合ともいう)の形式として，次のように表される．

$$\boldsymbol{x}=C_0^{(4)}\boldsymbol{\phi}^{(0)}+C_1^{(4)}\boldsymbol{\phi}^{(1)}+C_2^{(4)}\boldsymbol{\phi}^{(2)}+C_3^{(4)}\boldsymbol{\phi}^{(3)} \cdots\cdots\cdots(72)$$

式(72)のように表すことを，「直交展開する」といい，DCT 値 $\left\{C_k^{(4)}\right\}_{k=0}^{k=3}$ は「展開係数」(あるいは「DCT 係数」)とよばれる．この展開係数 $C_k^{(4)}$ を求めることが「信号の DCT 分解/合成」という物理的な意味を与えることになる(**図 16.5**)．

DCTとDFTとの関係

ところで，DFT から DCT への橋渡しに際し，式(22)に示すように，

(x_0-x_0)，(x_1-x_1)，(x_2-x_2)，(x_3-x_3)

と恒等的に‘0’となる信号を強制的に付加していることに気づかれたであろうか．もう少し話を進めてみることにしよう．

それでは，DFT を強く意識しながら，式(22)を書き直してみると，

$$\begin{aligned}X_{1/2}^{(4)}=&\frac{1}{8}\left\{x_0+x_1W_8+x_2W_8^{\ 2}+x_3W_8^{\ 3}\right.\\&\left.+x_3W_8^{\ 4}+x_2W_8^{\ 5}+x_1W_8^{\ 6}+x_0W_8^7\right\}\\&+\frac{1}{8}\left\{x_0+x_1W_8+x_2W_8^{\ 2}+x_3W_8^{\ 3}\right.\\&\left.-x_3W_8^{\ 4}-x_2W_8^{\ 5}-x_1W_8^{\ 6}-x_0W_8^7\right\}\end{aligned} \cdots\cdots\cdots(73)$$

となる．

右辺の第1項は8サンプルのディジタル信号，すなわち，

$\{x_0,\ x_1,\ x_2,\ x_3,\ x_3,\ x_2,\ x_1,\ x_0\}$

に対する DFT 値 $X_1^{(8)}$ であることがわかる(**図 16.6**)．つまり，DCT 値 $C_1^{(4)}$ は4サンプルの信号を対称に折り返した8サンプルの信号に対する DFT 値を計算することと等価であることを意味している．

また，右辺の第2項は，

$$\frac{1}{8}\left\{x_0+x_1W_8+x_2W_8^{\ 2}+x_3W_8^{\ 3}+(-x_3)W_8^{\ 4}+(-x_2)W_8^{\ 5}+(-x_1)W_8^{\ 6}+(-x_0)W_8^7\right\}$$

とみなすことができ，8サンプルのディジタル信号，すなわち，

$\{x_0,\ x_1,\ x_2,\ x_3,\ (-x_3),\ (-x_2),\ (-x_1),\ (-x_0)\}$

に対する DFT 値 $X_1^{(8)}$ であることがわかる(**図 16.7**)．別のいい方をすれば，DST 値 $S_1^{(4)}$ は4サンプルの信号 $\{x_0,\ x_1,\ x_2,\ x_3\}$ を反対称に折り返した8サンプル信号に対する DFT 値に相当することになるのである．

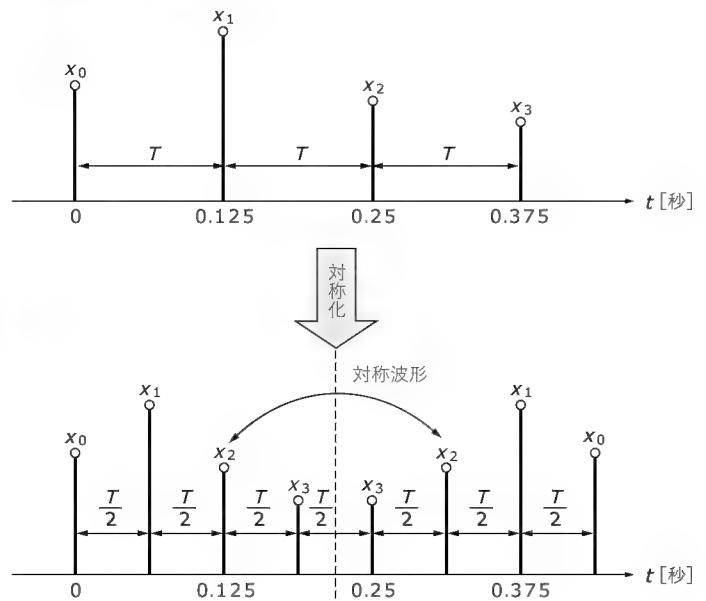
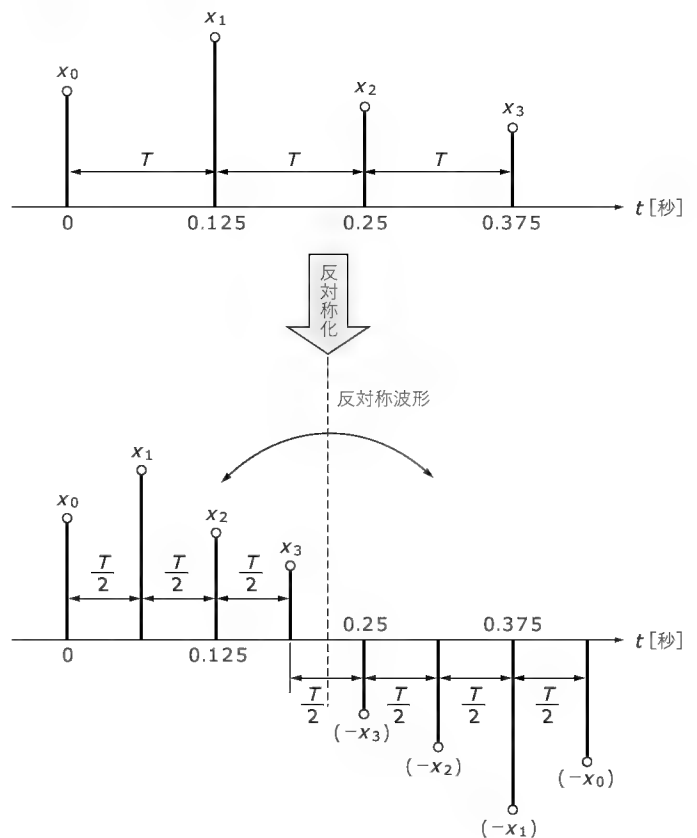
ここで，4サンプルに対する DCT と8サンプルに対する DFT の周波数の対応関係，すなわち，

$C_0^{(4)} \longleftrightarrow X_0^{(4)} \longleftrightarrow X_0^{(8)}$

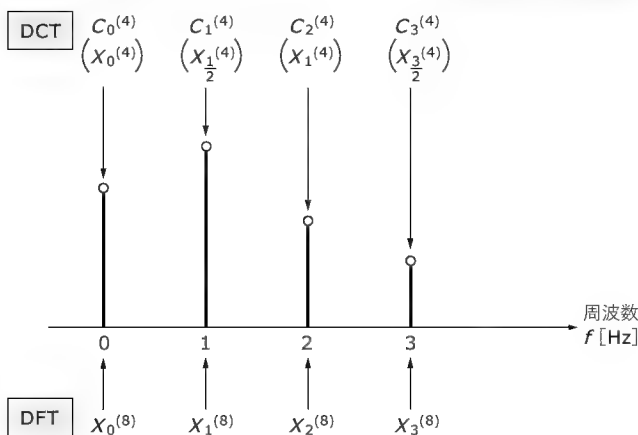
$C_1^{(4)} \longleftrightarrow X_{1/2}^{(4)} \longleftrightarrow X_1^{(8)}$

$C_2^{(4)} \longleftrightarrow X_1^{(4)} \longleftrightarrow X_2^{(8)}$

$C_3^{(4)} \longleftrightarrow X_{3/2}^{(4)} \longleftrightarrow X_3^{(8)}$

〔図 16.6〕DCT の考え方

〔図 16.7〕DST の考え方

となることも知っておいてもらいたい(**図 16.8**)．

なお，式(73)には，DCT 計算に DFT の高速計算アルゴリズム(FFT)を適用するためのヒントが隠されていることも，重要なポイントとしてしっかりと記憶に留めておいてほしい．

〔図16.8〕**DCT**(4サンプル)と**DFT**(8サンプル)の周波数の対応関係

つまるところ,

$X_0^{(4)} \rightarrow G_0^{(4)}$

$X_{1/2}^{(4)} \rightarrow G_1^{(4)}$

$X_1^{(4)} \rightarrow G_2^{(4)}$

$X_{3/2}^{(4)} \rightarrow G_3^{(4)}$

と表すことにすれば,DCT(あるいはDST)とDFTとの関係はℓ = 0,1,2,3として以下のようにまとめられる.

$$G^{(4)}=\frac{1}{\gamma}W_8^{-\frac{}{2}}\left[C^{(4)}-jS^{(4)}\right] \quad \cdots\cdots(74)$$

ただし,

$$C^{(4)}=\frac{1}{4}\sum_{n=0}^{3}\gamma\, x_n\cos\left\{\frac{(2n+1)}{8}\pi\right\} \quad \cdots\cdots(75)$$

$$S^{(4)}=\frac{1}{4}\sum_{n=0}^{3}\gamma\, x_n\sin\left\{\frac{(2n+1)}{8}\pi\right\} \quad \cdots\cdots(76)$$

$$\gamma=\begin{cases}1 & ;\ =0\\ \sqrt{2} & ;\ \neq 0\end{cases} \quad \cdots\cdots(77)$$

ここで,$C_\ell^{(4)}$,$S_\ell^{(4)}$はそれぞれ4サンプルのディジタル信号$\{x_0, x_1, x_2, x_3\}$に対するDCT値,DST値を表している.

例題3

2サンプルのディジタル信号$\{x_0, x_1\}$に対するDCTを求めよ.あわせて,正規直交基底ベクトル$\{\boldsymbol{\phi}^{(0)}, \boldsymbol{\phi}^{(1)}\}$を示せ.

解答3

まず,$\{x_0, x_1\}$のk = 0,1/2に対するDFT値はそれぞれ,

$$X_0^{(2)}=\frac{1}{2}\left(x_0+x_1\right)$$

$$\begin{aligned}X_{1/2}^{(2)}&=\frac{1}{2}\left\{x_0+x_1W_4\right\}\\&=\frac{1}{2}W_4^{-\frac{1}{2}}\left\{x_0W_4^{\frac{1}{2}}+x_1W_4^{\frac{3}{2}}\right\}\\&=\frac{1}{\sqrt{2}}W_4^{-\frac{1}{2}}\left[\frac{1}{2}\left\{\sqrt{2}x_0\cos\left(\frac{\pi}{4}\right)+\sqrt{2}x_1\cos\left(\frac{3\pi}{4}\right)\right\}\right.\\&\qquad\left.-j\frac{1}{2}\left\{\sqrt{2}x_0\sin\left(\frac{\pi}{4}\right)+\sqrt{2}x_1\sin\left(\frac{3\pi}{4}\right)\right\}\right]\end{aligned}$$

となる.したがって,DCTは実数部を採って,

$$C_0^{(2)}=\frac{1}{2}\left\{x_0+x_1\right\}$$

$$C_1^{(2)}=\frac{1}{2}\left\{\sqrt{2}x_0\cos\left(\frac{\pi}{4}\right)+\sqrt{2}x_1\cos\left(\frac{3\pi}{4}\right)\right\}=\frac{1}{2}\left\{x_0-x_1\right\}$$

となる.よって,$C_0^{(2)}$と$C_1^{(2)}$の{ }の中の係数を並べることにより,正規直交基底ベクトルが以下のように得られる.

$$\boldsymbol{\phi}^{(0)}=\left\{1,\ 1\right\},$$

$$\boldsymbol{\phi}^{(1)}=\left\{\sqrt{2}\cos\left(\frac{\pi}{4}\right),\ \sqrt{2}\cos\left(\frac{3\pi}{4}\right)\right\}=\left\{1,\ -1\right\}$$

このN = 2に対するDCTの正規直交基底ベクトルは,「ハール変換(Haar Transform)基底」,あるいは「ウォルシュ型のアダマール変換(Walsh-Hadamard Transform)の基底」とよばれる.

*　　*

次回は,DCTの物理的な意味付け,一般式を示すとともに,本格的に“実務に直結して応用できる”DCTによる信号解析の基礎について解説する予定である.お楽しみに.

みたに・まさあき　東京電機大学工学部情報通信工学科

x86CPUだけでもマスタしたい

開発技術者のためのアセンブラ入門

第18回　ビット/バイト命令とフラグ制御命令　大貫広幸

前回に引き続き，今回もx86系CPUがもっているビットやフラグの操作に関する命令について説明します．

前回は，ビット操作に関する命令としては，AND，OR，NOTといったビット単位の論理演算を行う「論理命令」と，指定ビット数分のシフトや回転を行う「シフト命令」，「ローテート命令」について説明しました．

今回は，ビットやフラグの操作に関する命令のうち，前回説明できなかったビットのテストやセット/リセットを行う「ビット命令」と，ステータスフラグの状態をバイト値として取得するための「バイト命令」，そしてフラグのセット/リセット，ロード/ストアを行う「フラグ制御命令」について説明します．

ビット命令とバイト命令

ビット命令は，指定ビットの状態を調べたり，セット/リセット/反転といった操作を行う命令です(**表1**)．

ビット命令としては，BT，BTS，BTR，BTC，BSF，BSR，TESTの七つの命令があります．TEST命令については前回，すでに説明したので，ここでは残りのBで始まる六つの命令について説明します．

バイト命令は，ステータスフラグ(OF，SF，ZF，AF，PF，CF)の状態をバイト値としてストアするための命令です．バイト命令には，SETcc命令があります．

● BT，BTS，BTR，BTC命令

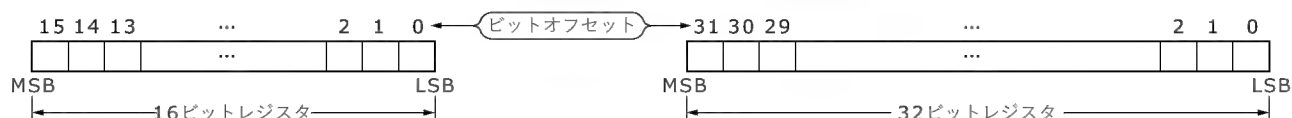
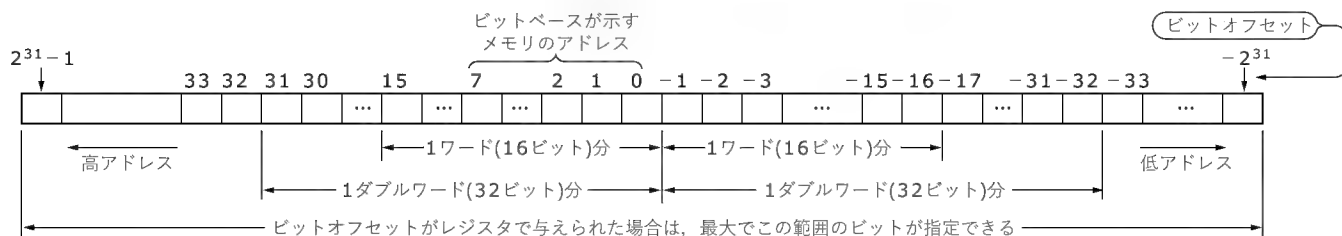
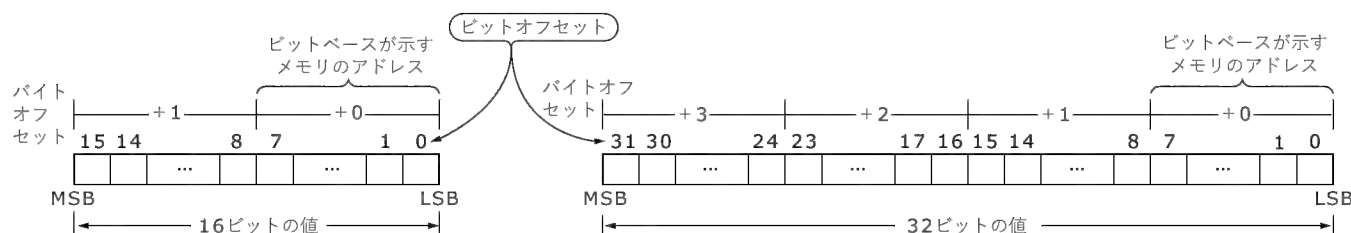
このBで始まる四つの命令は，ビットベースとビットオフセットという二つのオペランドをもちます．オペランドの位置関係は，ビットベースが転送先(DEST)，ビットオフセットが転送元(SOU)の位置となります．

ビットベースは，操作対象のビットがある16ビット長のワード整数，および32ビット長のダブルワード整数を格納している汎用レジスタ，あるいはメモリ上の配列のベースアドレスを指定します．

〔表1〕x86系の32ビットCPUで使用できるビット命令とバイト命令

分類	インストラクション名	動作	影響を受けるフラグ					
			OF	SF	ZF	AF	PF	CF
ビット命令	BT	Bit Test ビットベースで指定されたビット列のビットオフセットの位置にあるビットの値をCFにコピーする	?	?	?	?	?	*
	BTS	Bit Test and Set ビットベースで指定されたビット列のビットオフセットの位置にあるビットの値をCFにコピーした後，そのビットをセット(1)する	?	?	?	?	?	*
	BTR	Bit Test and Reset ビットベースで指定されたビット列のビットオフセットの位置にあるビットの値をCFにコピーした後，そのビットをリセット(0)する	?	?	?	?	?	*
	BTC	Bit Test and Complement ビットベースで指定されたビット列のビットオフセットの位置にあるビットの値をCFにコピーした後，そのビットを反転(0→1，1→0)する	?	?	?	?	?	*
	BSF	Bit Scan Forward SOUを最下位ビットから上位ビットの方向に1になっているビットを探し，ビットインデックスをDESTに格納する	?	?	*	?	?	?
	BSR	Bit Scan Reverse SOUを最上位ビットから下位ビットの方向に1になっているビットを探し，ビットインデックスをDESTに格納する	?	?	*	?	?	?
バイト命令	SETcc	Set Byte on Condition ccで示された条件が成立する場合はDEST←1 成立しない場合はDEST←0	·	·	·	·	·	·

●表中のDESTはdestination(先)，SOUはsource(元)を表す
●表中の影響を受けるフラグの記号は次の状態を表す　?＝未定義　・＝変化ない　*＝結果にしたがって変化する

〔図1〕**BT，BTS，BTR，BTC 命令でビットベースがレジスタの場合のビットオフセットの範囲**

〔図2〕**BT，BTS，BTR，BTC 命令でビットベースがメモリの場合のビットオフセットの範囲**

(**a**) ビットオフセットをレジスタで指定した場合

(**b**) ビットオフセットをイミディエイトで指定し，メモリ上のビットを16ビット値としてアクセスする場合

(**c**) ビットオフセットをイミディエイトで指定し，メモリ上のビットを32ビット値としてアクセスする場合

〔リスト1〕**MASM の BT，BTS，BTR，BTC，BSF，BSR 命令の記述例**

```
                                .586
                                .model flat

00000000                        .data

00000000  01                     dtByte  db      1
00000001  0002                   dtWord  dw      2
00000003  00000004               dtDWord dd      4

00000000                        .code
00000000  66| 0F A3 F0                   bt      ax,si
00000004  0F A3 DA                       bt      edx,ebx
00000007  66, 0F A3 0D                   bt      dtWord,cx
          00000001 R
0000000F  0F A3 15                       bt      dtDWord,edx
          00000003 R
00000016  66| 0F BA E0                   bt      ax,3
          03
0000001B  0F BA E2 1B                    bt      edx,27
0000001F  66| 0F BA 25                   bt      dtWord,13
          00000001 R 0D
00000028  0F BA 25                       bt      dtDWord,15
          00000003 R 0F
                                ;
00000030  66| 0F AB F0                   bts     ax,si
00000034  0F AB DA                       bts     edx,ebx
00000037  66, 0F AB 0D                   bts     dtWord,cx
          00000001 R
0000003F  0F AB 15                       bts     dtDWord,edx
          00000003 R
00000046  66  0F BA E8                   bts     ax,3
          03
0000004B  0F BA EA 1B                    bts     edx,27
0000004F  66| 0F BA 2D                   bts     dtWord,13
          00000001 R 0D
00000058  0F BA 2D                       bts     dtDWord,15
          00000003 R 0F
                                ;
00000060  66| 0F B3 F0                   btr     ax,si
00000064  0F B3 DA                       btr     edx,ebx
00000067  66| 0F B3 0D                   btr     dtWord,cx
          00000001 R
0000006F  0F B3 15                       btr     dtDWord,edx
          00000003 R
00000076  66| 0F BA F0                   btr     ax,3
          03
0000007B  0F BA F2 1B                    btr     edx,27
0000007F  66| 0F BA 35                   btr     dtWord,13
          00000001 R 0D
00000088  0F BA 35                       btr     dtDWord,15
          00000003 R 0F
                                ;
00000090  66| 0F BB F0                   btc     ax,si
00000094  0F BB DA                       btc     edx,ebx
00000097  66| 0F BB 0D                   btc     dtWord,cx
          00000001 R
0000009F  0F BB 15                       btc     dtDWord,edx
          00000003 R
000000A6  66| 0F BA F8                   btc     ax,3
          03
000000AB  0F BA FA 1B                    btc     edx,27
000000AF  66| 0F BA 3D                   btc     dtWord,13
          00000001 R 0D
000000B8  0F BA 3D                       btc     dtDWord,15
          00000003 R 0F
                                ;
                                ;-------------------------------
                                ;
000000C0  66| 0F BC D6                   bsf     dx,si
000000C4  0F BC C3                       bsf     eax,ebx
000000C7  66| 0F BC 3D                   bsf     di,dtWord
          00000001 R
000000CF  0F BC 0D                       bsf     ecx,dtDWord
          00000003 R
                                ;
000000D6  66| 0F BD D6                   bsr     dx,si
000000DA  0F BD C3                       bsr     eax,ebx
000000DD  66| 0F BD 3D                   bsr     di,dtWord
          00000001 R
000000E5  0F BD 0D                       bsr     ecx,dtDWord
          00000003 R
                                end
```

ビットベース
ビットオフセット

ビットオフセットは，ビットベースが示すレジスタ，あるいはメモリのビット0を，ゼロとするオフセットで，操作対象のビットを指定します．この場合，オフセットはプラス方向が最下位ビット(LSB)から最上位ビット(MSB)への方向で，アドレスは増加する方向となります．逆に，マイナス方向のオフセットは最上位ビット(MSB)から最下位ビット(LSB)への方向で，アドレスは減少する方向となります．ビットオフセットとしては，ビットベースと同じオペランドサイズの汎用レジスタか，バイトのイミディエイト値を指定します．

ビットベースがレジスタの場合，ビットオフセット値は，ビットベースのレジスタのサイズ(16あるいは32)で剰余をとった値の範囲のビットを対象とします．そのため，ビットベースのレジスタが16ビットなら0～15，32ビットなら0～31の範囲のビットが操作対象のビットとなります(**図1**)．

ビットベースでメモリを指定した場合，ビットオフセットが汎用レジスタかイミディエイトかでビットの範囲が異なります．

ビットオフセットが汎用レジスタで指定されているのなら，符号付きの16ビット，あるいは32ビット整数で表せる範囲内のビット位置を指定できます〔**図2(a)**〕．しかし，ビットオフセットにイミディエイトを指定した場合は，16ビットでメモリをアクセスする場合，下位4ビットのみが使用されるので0～15の範囲のビット，そして32ビットでメモリをアクセスする場合，下位5ビットのみが使用されるので0～31の範囲のビットしか指定できません〔**図2(b)**および**図2(c)**〕．

メモリに対するBT，BTS，BTR，BTC命令は，指定された16ビットあるいは32ビット，つまりワードあるいはダブルワードの単位でメモリをアクセスします．そのため，バイトのポートが接続されたメモリマップドI/Oで，このBT，BTS，BTR，BTC命令を使用する場合には，注意する必要があります．

メモリマップドI/Oで接続されたバイトポートに対してBT，BTS，BTR，BTC命令を実行すると，連続する別のバイトポートも同時にアクセスされることになります．そのため，このようなアクセスが別のバイトポートに影響を与える場合もあるので，バイトポートに対するBT，BTS，BTR，BTC命令の使用には注意が必要になるわけです．

BT，BTS，BTR，BTCの各命令の実際のMASMでの記述例を**リスト1**，gasでの記述例を**リスト2**に示します．また，BT，BTS，BTR，BTCの各命令の動作は次のようになっています．

(1) BT命令

ビットベースとビットオフセットで指定されたビットの値(0/1)をCFにコピーします．CF以外のステータスフラグ(OF，SF，ZF，AF，PF)は未定義となります．

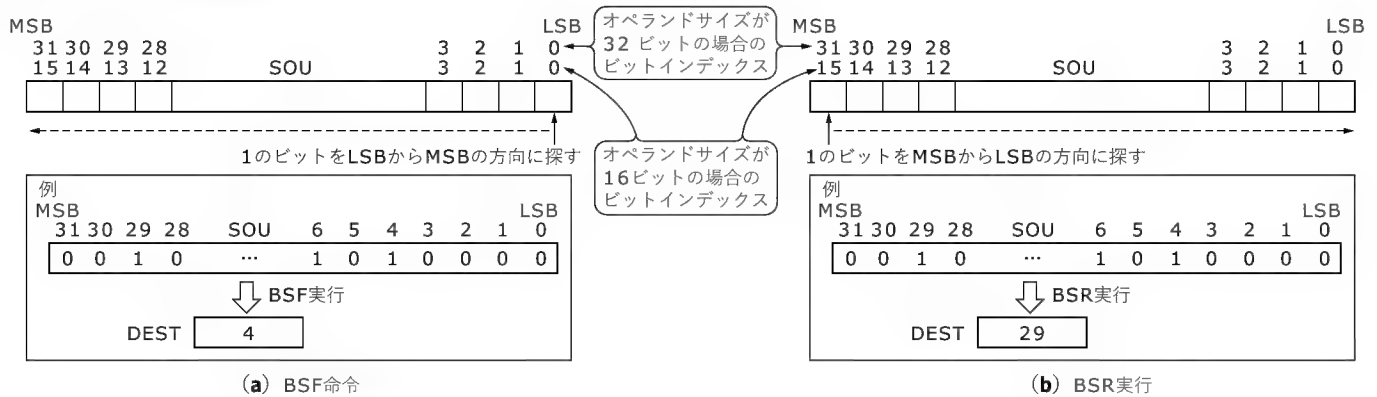
(2) BTS命令

まず，ビットベースとビットオフセットで指定されたビットの

〔リスト2〕gasのBT，BTS，BTR，BTC，BSF，BSR命令の記述例

```
 1                    .data
 2
 3  0000 01           dtByte:    .byte  1   ← ビットオフセット
 4  0001 0200         dtWord:    .word  2
 5  0003 04000000     dtDWord:   .long  4   ← ビットベース
 6
 7                    .text
 8  0000 660FA3F0             btw     %si,%ax
 9  0004 0FA3DA               btl     %ebx,%edx
10  0007 660FA30D             btw     %cx,dtWord
10       01000000
11  000f 0FA31503             btl     %edx,dtDWord
11       000000
12  0016 660FBAE0             btw     $3,%ax
12       03
13  001b 0FBAE21B             btl     $27,%edx
14  001f 660FBA25             btw     $13,dtWord
14       01000000
14       0D
15  0028 0FBA2503             btl     $15,dtDWord
15       0000000F
16                    #
17  0030 660FABF0             btsw    %si,%ax
18  0034 0FABDA               btsl    %ebx,%edx
19  0037 660FAB0D             btsw    %cx,dtWord
19       01000000
20  003f 0FAB1503             btsl    %edx,dtDWord
20       000000
21  0046 660FBAE8             btsw    $3,%ax
21       03
22  004b 0FBAEA1B             btsl    $27,%edx
23  004f 660FBA2D             btsw    $13,dtWord
23       01000000
23       0D
24  0058 0FBA2D03             btsl    $15,dtDWord
24       0000000F
25                    #
26  0060 660FB3F0             btrw    %si,%ax
27  0064 0FB3DA               btrl    %ebx,%edx
28  0067 660FB30D             btrw    %cx,dtWord
28       01000000
29  006f 0FB31503             btrl    %edx,dtDWord
29       000000
30  0076 660FBAF0             btrw    $3,%ax
30       03
31  007b 0FBAF21B             btrl    $27,%edx
32  007f 660FBA35             btrw    $13,dtWord
32       01000000
32       0D
33  0088 0FBA3503             btrl    $15,dtDWord
33       0000000F
34                    #
35  0090 660FBBF0             btcw    %si,%ax
36  0094 0FBBDA               btcl    %ebx,%edx
37  0097 660FBB0D             btcw    %cx,dtWord
37       01000000
38  009f 0FBB1503             btcl    %edx,dtDWord
38       000000
39  00a6 660FBAF8             btcw    $3,%ax
39       03
40  00ab 0FBAFA1B             btcl    $27,%edx
41  00af 660FBA3D             btcw    $13,dtWord
41       01000000
41       0D
42  00b8 0FBA3D03             btcl    $15,dtDWord
42       0000000F
43                    #
44                    #------------------------------------
45                    #
46  00c0 660FBCD6             bsfw    %si,%dx
47  00c4 0FBCC3               bsfl    %ebx,%eax
48  00c7 660FBC3D             bsfw    dtWord,%di
48       01000000
49  00cf 0FBC0D03             bsfl    dtDWord,%ecx
49       000000
50                    #
51  00d6 660FBDD6             bsrw    %si,%dx
52  00da 0FBDC3               bsrl    %ebx,%eax
53  00dd 660FBD3D             bsrw    dtWord,%di
53       01000000
54  00e5 0FBD0D03             bsrl    dtDWord,%ecx
54       000000
```

〔図3〕BSF, BSR命令の動作

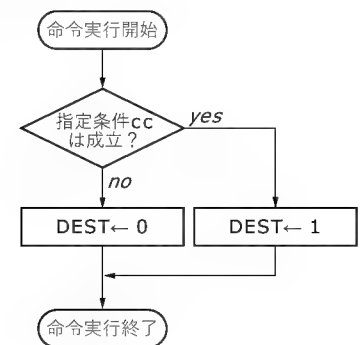
(a) BSF命令　　(b) BSR実行

〔表2〕SETcc命令のニモニックで使われる条件を表す文字

文字	内 容	成立となる条件
E	Equal ……等しい	ZF = 1
Z	Zero ……ゼロ	ZF = 1
NE	Not Equal ……等しくない	ZF = 0
NZ	Not Zero ……ゼロではない	ZF = 0
A	Above ……より上	CF = 0 and ZF = 0
NBE	Not Below or Equal ……より下でなく等しくない	CF = 0 and ZF = 0
AE	Above or Equal ……より上か等しい	CF = 0
NB	Not Below ……より下でない	CF = 0
B	Below ……より下	CF = 1
NAE	Not Above or Equal ……より上でなく等しくない	CF = 1
BE	Below or Equal ……より下か等しい	CF = 1 or ZF = 1
NA	Not Above ……より上でない	CF = 1 or ZF = 1
G	Greater ……より大きい	ZF = 0 and SF = OF
NLE	Not Less or Equal ……より小さくなく等しくない	ZF = 0 and SF = OF
GE	Greater or Equal ……より大きいか等しい	SF = OF
NL	Not Less ……より小さくない	SF = OF
L	Less ……より小さい	SF ≠ OF
NGE	Not Greater or Equal ……より大きくなく等しくない	SF ≠ OF
LE	Less or Equal ……より小さいか等しい	ZF = 1 or SF ≠ OF
NG	Not Greater ……より大きくない	ZF = 1 or SF ≠ OF
C	Carry ……キャリーがある	CF = 1
NC	Not Carry ……キャリーがない	CF = 0
O	Overflow ……オーバフローがある	OF = 1
NO	Not Overflow ……オーバフローがない	OF = 0
S	Sign ……符号がある(負数)	SF = 1
NS	Not Sign ……符号がない(非負数)	SF = 0
P	Parity ……パリティがある	PF = 1
PE	Parity Even ……パリティが偶数	PF = 1
NP	Not Parity ……パリティがない	PF = 0
PO	Parity Odd ……パリティが奇数	PF = 0

値(0/1)をCFにコピーした後，そのビットを1にセットします．CF以外のステータスフラグ(OF, SF, ZF, AF, PF)は未定義となります．

(3) BTR命令

まず，ビットベースとビットオフセットで指定されたビットの

〔図4〕SETcc命令の動作

値(0/1)をCFにコピーした後，そのビットを0にリセットします．CF以外のステータスフラグ(OF, SF, ZF, AF, PF)は未定義となります．

(4) BTC命令

まず，ビットベースとビットオフセットで指定されたビットの値(0/1)をCFにコピーした後，そのビットを反転(0なら1, 0なら1に)します．CF以外のステータスフラグ(OF, SF, ZF, AF, PF)は未定義となります．

● BSF，BSR命令

この命令は，転送元(SOU)で指定された汎用レジスタあるいはメモリ上の値を，BSFでは下位から上位に向かって，BSRでは上位から下位に向かって1になっているビットを探すというものです(**図3**).

ビット位置(ビットインデックス)は最下位ビットをゼロとし，上位に向かって1，2，3，……となります．

転送先(DEST)には，1になっいるビットの位置が格納されます．ただし，転送元(SOU)に1なっているビットがなければ(つまり値がゼロ)なら，転送先(DEST)の内容は未定義となります．

転送元(SOU)には，16ビットのワード整数および32ビットのダブルワード整数が指定できます．BSF，BSR命令実行により ZFを除くステータスフラグ(OF, SF, AF, PF, CF)は未定義となります．ZFは，転送元(SOU)がゼロだった場合に ZF = 1，そうでなければ ZF = 0となります．

実際のMASMでのBSF，BSR命令の記述例を**リスト1**，gas

〔リスト3〕MASMのSETcc命令の記述例

```
                              .586
                              .model flat

00000000                      .data

00000000 01                   dtByte  db      1

00000000                      .code
00000000  0F 94 C2                    sete    dl
00000003  0F 94 05                    sete    dtByte
          00000000 R
0000000A  0F 95 C2                    setne   dl
0000000D  0F 95 05                    setne   dtByte
          00000000 R

00000014  0F 92 C0                    setc    al
00000017  0F 92 05                    setc    dtByte
          00000000 R
0000001E  0F 93 C0                    setnc   al
00000021  0F 93 05                    setnc   dtByte
          00000000 R

00000028  0F 9F C5                    setg    ch
0000002B  0F 9F C5                    setnle  ch
0000002E  0F 9F 05                    setg    dtByte
          00000000 R
00000035  0F 9F 05                    setnle  dtByte
          00000000 R
                              end
```

SETcc命令のcc条件は**表2**のように30種類あるが，この例では上からE(等しい)，NE(等しくない)，C(CF=1)，NC(CF=0)，G(より大きい)，NLE(より小さくなく等しくない)を指定している

〔リスト4〕gasのSETcc命令の記述例

```
 1                    .data
 2
 3  0000 01           dtByte:   .byte  1
 4
 5                    .text
 6  0000 0F94C2              sete    %dl
 7  0003 0F940500            sete    dtByte
 7       000000
 8
 9  000a 0F95C2              setne   %dl
10  000d 0F950500            setne   dtByte
10       000000
11
12  0014 0F92C0              setc    %al
13  0017 0F920500            setc    dtByte
13       000000
14
15  001e 0F93C0              setnc   %al
16  0021 0F930500            setnc   dtByte
16       000000
17
18  0028 0F9FC5              setg    %ch
19  002b 0F9FC5              setnle  %ch
20  002e 0F9F0500            setg    dtByte
20       000000
21  0035 0F9F0500            setnle  dtByte
21       000000
22
23  003c 0F94C0              setzb   %al
24  003f 0F94C0              setz    %al
25  0042 0F940500            setzb   dtByte
25       000000
26  0049 0F940500            setz    dtByte
26       000000
```

SETcc命令ではオペランドはバイトしかないため，オペランドのサイズを表すbはとくに付けなくてもよい

での記述例を**リスト2**に示します．

● SETcc命令

SETcc命令は，ccで示されたステータスフラグ(OF，SF，ZF，AF，PF，CF)の状態を，バイト値としてストアするための命令です．SETcc命令のccの部分には，**表2**に示す条件を表す文字が入ります．

SETcc命令でストアされるバイト値は，指定条件ccの状態が成立していれば1，成立していなければ0がストアされます(**図4**)．SETcc命令自体は，実行してもステータスフラグには影響を与えません．SETcc命令には，転送先(DEST)なる8ビット長のレジスタ，あるいはメモリを示すオペランドを一つ指定します．

実際のMASMでのSETcc命令の記述例を**リスト3**，gasでの記述例を**リスト4**に示します．

フラグ制御命令

フラグ制御命令は，レジスタEFLAGS内のフラグを操作するための命令で，STC，CLC，CMC，STD，CLD，STI，CLI，PUSHF(D)，POPF(D)，LAHF，SAHFの11命令あります(**表3**)．この11あるフラグ制御命令は，オペランドをもちません．そのため，アセンブラにはニモニックのみを記述します．

実際のMASMでの記述例を**リスト5**，gasでの記述例を**リスト6**に示します．

● STC，CLC，CMC命令

この3命令はCFを操作するための命令です．STCがCF＝1，CLCがCF＝0，CMCがCFの値を反転する命令です．この三つの命令は，CFを指定の状態に変化させるのみで，他のフラグには影響を与えません．

● STD，CLD命令

ストリング命令で使用されるフラグDFを操作する命令です．STDがDF＝1，CLDがDF＝0にする命令です．この二つの命令はDFを指定の状態に変化させるのみで，他のフラグには影響を与えません．

● STI，CLI命令

レジスタEFLAGS上の割り込み許可フラグIFを操作する命令です．STIがIF＝1，CLIがIF＝0にする命令です．この二つの命令は，IFを指定の状態に変化させるのみで，他のフラグには影響を与えません．

CPUは，IF＝1の状態のとき外部からのハードウェア割り込みを受け付けます．逆にIF＝0の状態のときは，外部からのハードウェア割り込みを禁止します．つまり，STIは割り込み許可，CLIが割り込み禁止の命令となります．

STI，CLI命令は，CPUの動作モードにより使用できない場合があります．CPUがプロテクトモードで，プログラムの特権レベルが低い場合には，CPUの保護機能によりSTI，CLI命令は使用できなくなっています．もし，この状態でSTI，CLI命

令を実行した場合は，例外発生となり，例外処理ルーチンへのジャンプとなります．

通常，Windows や Linux 上で実行されているアプリケーションプログラムは，低い特権レベルで CPU が実行されています．そのため，Windows や Linux 上で実行するアプリケーションプログラムでは，この STI，CLI 命令は使用しません．

● PUSHF(D)命令

PUSHF(D)命令は，レジスタ EFLAGS の内容をスタックに PUSH する命令です．

レジスタ EFLAGS の下位 16 ビット(FLAGS)を PUSH する場合は，PUSHF 命令を使用します．そして，レジスタ EFLAGS 全体を PUSH する場合は，PUSHFD 命令を使用します．この PUSHF(D)命令を実行してもレジスタ EFLAGS の内容は変化しません．

● POPF(D)命令

POPF(D)命令は，スタックに退避されていたレジスタ EFLAGS の内容をスタックから POP する命令です．

レジスタ EFLAGS の下位 16 ビット(FLAGS)を POP する場合は，POPF 命令を使用します．そして，レジスタ EFLAGS 全体を

〔表 3〕x86 系の 32 ビット CPU で使用できるフラグ制御命令

分類	インストラクション名	動 作	影響を受けるフラグ							
			OF	DF	IF	SF	ZF	AF	PF	CF
フラグ制御命令	STC	Set Carry Flag レジスタ EFLAGS の CF をセット(1)する	·	·	·	·	·	·	·	1
	CLC	Clear Carry Flag レジスタ EFLAGS の CF をクリア(0)する	·	·	·	·	·	·	·	0
	CMC	Complement Carry Flag レジスタ EFLAGS の CF を反転(0→1，1→0)する	·	·	·	·	·	·	·	*
	STD	Set Direction Flag レジスタ EFLAGS の DF をセット(1)する	·	1	·	·	·	·	·	·
	CLD	Clear Direction Flag レジスタ EFLAGS の DF をクリア(0)する	·	0	·	·	·	·	·	·
	STI	Set Interrupt Flag レジスタ EFLAGS の IF をセット(1)する	·	·	1	·	·	·	·	·
	CLI	Clear Interrupt Flag レジスタ EFLAGS の IF をクリア(0)する	·	·	0	·	·	·	·	·
	PUSHF(D)	Push EFLAGS onto Stack レジスタ EFLAGS の内容をスタックに PUSH する	·	·	·	·	·	·	·	·
	POPF(D)	Pop EFLAGS from Stack スタックからレジスタ EFLAGS の内容を POP する	*	*	*	*	*	*	*	*
	LAHF	Load Flags into AH Register レジスタ EFLAGS の下位 8 ビットをレジスタ AH に転送する	·	·	·	·	·	·	·	·
	SAHF	Store AH Register into Flags レジスタ AH の値をレジスタ EFLAGS の下位 8 ビットに転送する	·	·	·	*	*	*	*	*

●表中の影響を受けるフラグの記号は次の状態を表す
・＝変化ない
*＝結果にしたがって変化する
0＝クリアされる
1＝セットされる

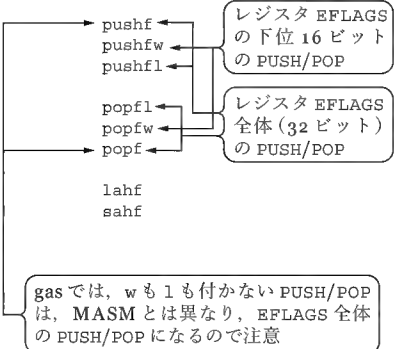
〔リスト 5〕MASM のフラグ制御命令の記述例

```
                                .586
                                .model flat

00000000                        .code
00000000  F9                            stc
00000001  F8                            clc
00000002  F5                            cmc

00000003  FD                            std
00000004  FC                            cld

00000005  FB                            sti
00000006  FA                            cli

00000007  66| 9C                        pushf
00000009  9C                            pushfd

0000000A  9D                            popfd
0000000B  66| 9D                        popf

0000000D  9F                            lahf
0000000E  9E                            sahf
                                end
```

レジスタ EFLAGS の下位 16 ビットの PUSH/POP

レジスタ EFLAGS 全体(32 ビット)PUSH/POP

〔リスト 6〕gas のフラグ制御命令の記述例

```
 1                      .text
 2  0000 F9                     stc
 3  0001 F8                     clc
 4  0002 F5                     cmc
 5
 6  0003 FD                     std
 7  0004 FC                     cld
 8
 9  0005 FB                     sti
10  0006 FA                     cli
11
12  0007 9C                     pushf
13  0008 669C                   pushfw
14  000a 9C                     pushfl
15
16  000b 9D                     popfl
17  000c 669D                   popfw
18  000e 9D                     popf
19
20  000f 9F                     lahf
21  0010 9E                     sahf
22
```

POPする場合は，POPFD命令を使用します．このPOPF(D)命令を実行するとレジスタEFLAGSの内容はPOPした内容に書き換えられます．ただし，CPUの動作モードにより保護されるフラグがあるため，保護されているフラグはPOPF(D)命令を実行しても変化しません．

● LAHF，SAHF命令

LAHF命令は，レジスタEFLAGSの下位8ビット(SF，ZF，0，AF，0，PF，1，CF)をレジスタAHに転送する命令です．LAHF命令を実行しても，レジスタEFLAGSには影響を与えません．

SAHF命令は，レジスタAHの値をレジスタEFLAGSの下位8ビットに転送する命令です．実際に影響を受けるフラグはレジスタEFLAGSのSF，ZF，AF，PF，CFのみです(**図5**)．

＊　＊

次回は，x86系CPUのプログラムの実行制御に関する命令について説明する予定です．

〔図5〕LAHF，SAHF命令の動作

レジスタEFLAGS
31　7 6 5 4 3 2 1 0
SF ZF 0 AF 0 PF 1 CF
ビットがコピーされる
7　レジスタAH　0

(a) LAHF命令

7　レジスタAH　0
□で示したビットのみがコピーされる
SF ZF 0 AF 0 PF 1 CF
31　7 6 5 4 3 2 1 0
レジスタEFLAGS

(b) SAHF命令

おおぬき・ひろゆき　大貫ソフトウェア設計事務所

第9回

C99規格についての説明と検証

岸 哲夫

「ISO/IEC 9899 : 1999 - Programming Language C」(略称： C99)規格は1999年12月にISOによって規格化された新機能である．現在はデフォルトで，C99規格で決定した新機能の一部を使用することができる．また，新機能の一部はGNU Cの拡張機能と同一である場合もある．今回はGNU CのC99への対応について，解説とテストを行う．(編集部)

前回までで，GCCにおける拡張機能についての説明を終わりました．

今回は，「ISO/IEC 9899 : 1999 - Programming Language C」(略称： C99)規格について説明と検証をします．

連載の第3回(2002年10月号)で少し触れましたが，“-flang-isoc9x”というオプションを指定してコンパイルすることで，このC99規格で決定した新機能の一部を使用することができるはずでしたが，オプション“-flang-isoc9x”は廃止されました．

現在は，デフォルトでC99規格で決定した新機能の一部を使用することができます．

なお，新機能の一部がGNU Cの拡張機能と同一である場合もあります．

また後述しますが，GCCのバージョンによっては，一部の機能でライブラリが対応していなかったり，バグがあるものや実装されていないものもあります．

C99規格の予約語について

追加された予約語は，inline，restrict，_Bool，_Complex，_Imaginaryです．

● inline

従来はC++の予約語でした．第六回で説明したように，C++では使用できてもCでは使用できません．今回は予約語になりましたが，理由があってGNU Cでは機能していません．

-ansiオプションでコンパイルすると，inlineという単語を変数として使用できますが，-ansiオプションがないとC99規格をサポートするので，エラーになります．

〔リスト1〕単語inlineを変数としてテスト(test106.c)

```
#include <stdio.h>
/*inlineは予約語?*/
int main(void)
{
    int  inline;
    return;
}
```

ただし，inline関数を指定しても，実質は何もしません．以下は，**リスト1**のコンパイル結果です．

```
$ gcc    -S  -ansi test106.c
$
$ gcc    -S   test106.c
test106.c: In function `main':
test106.c:5: warning: useless keyword or
                type name in empty declaration
test106.c:5: warning: empty declaration
$
```

● restrict

ポインタの修飾に使用します．そのポインタが別名定義されていない場合に，そのことを明確にコンパイラに知らせるために使用します．

● _Bool

値0か1を保存できるブール変数の型です．

● _Complex

複素数を扱うために使用します．

● _Imaginary

複素数の虚数部だけを扱うために使用します．

新しく定義されたマクロとプラグマ

● マクロ

__STDC_VERSION__，__STDC_ISO_10646__，__STDC_IEC_559__，__STDC_IEC_559_COMPLEX__の四つの定義済みマクロが使用できます．

詳しくは，GNU Cのプリプロセッサの項で詳細に説明します．

● プラグマ

以下のプラグマ(pragma)が追加されました．

● #pragma STDC FP_CONTRACT on-off-switch

浮動小数点演算の計算方法を指定します．

● #pragma STDC FENV_ACCESS on-off-switch

浮動小数点演算を使用するかどうかを指定します．

● #pragma STDC CX_LIMITED_RANGE on-off-switch

複素数計算をするときに指定します.

詳しくは, GNU Cのプリプロセッサの項で詳細に説明します.

● _Pragmaという単項演算子

プラグマは処理系独自のものなので, 標準化を行うことはできません. しかし, 可搬性を高めるため,

```
" #define pragma_FP_CONTRACT _Pragma
      ("STDC FP_CONTRACT on-off-switch")"
```

と指定できるようになりました.

この#define文を#if文で制御すれば, 環境ごとに対応させることによる#if文の指定も最小限で済みます.

また,

```
#if defined(FP_ENABLE)
 #define pragma_FP_CONTRACT _Pragma
          ("STDC FP_CONTRACT on-off-switch")
```

のように指定すればわかりやすくなります.

C99規格のマクロ

● 関数型マクロ呼び出しの際の空引き数

ANSI Cでは, マクロ呼び出しの際に引き数が必要ないからといって, 空の引き数を使用することはできませんでした. しかしC99規格では, そのような呼び出しを許します.

ただ, GNU Cでは-ansiをつけても, -traditionalをつけてコンパイルしてもエラーにはなりませんでした.

リスト2のコンパイルと実行結果を以下に示します.

```
$ gcc -ansi test107.c -o test107
$ ./test107
エラーコード100です 詳細はerr01
エラーコード200です 詳細は
$ gcc -traditional test107.c -o test107
$ ./test107
エラーコード100です 詳細はerr01
エラーコード200です 詳細は
$ gcc test107.c -o test107
$ ./test107
エラーコード100です 詳細はerr01
エラーコード200です 詳細は
$
```

● 可変個数の引き数をもつマクロ

連載第7回(2003年3月号)で, 拡張機能として“可変個数の引き数をもつマクロ”の説明をしましたが, 同じことをC99規格でも実現しています. ただし, -ansiオプションをつけるとエラーになります.

リスト3のコンパイルと実行結果を次に示します.

〔リスト2〕関数型マクロ呼び出しの際の空引き数(test107.c)

```
#define message(errcd,msg) "エラーコード"  errcd  "です 詳細は"  msg
main()
{
    printf("%s¥n",message("100","err01"));
    printf("%s¥n",message("200",));
}
```

〔リスト3〕可変個数の引き数を持つマクロ(test108.c)

```
/*
 * 可変個数の引数を持つマクロの例
 */
#include <stdio.h>
#define M_debug(format, ...)    printf("debug:" format, __VA_ARGS__)
main()
{
    int  x01  =    100;
    int  x02  =    200;
    M_debug("x01=%d¥n",x01);
    M_debug("(%d,%d)¥n",x01,x02);
    M_debug("(%d,%d,%d行目)¥n",x01,x02,__LINE__ );
}
```

```
$ gcc -ansi -pedantic test108.c -o test108
test108.c:5: badly punctuated parameter
                              list in `#define'
$
$ gcc test108.c -o test108
$ ./test108
debug:x01=100
debug:(100,200)
debug:(100,200,13行目)
$
```

その他のプリプロセッサ関連

● プリプロセッサ式中の整数型について

プリプロセッサ内部で演算をする際, 従来はlongとunsigned longでしか扱えませんでした.

C99規格では, 新しく導入されたヘッダ“stdint.h”にあるように, intmax_t型, uintmax_t型が使用できるようになりました. インテルアーキテクチャのGCC 2.95では, 以下のように定義されています.

```
$ cat /usr/include/stdint.h | grep intmax_t
typedef long int                intmax_t;
typedef unsigned long int       uintmax_t;
typedef long long int           intmax_t;
typedef unsigned long long int  uintmax_t;
$
```

● 文字列定数とワイド文字列定数を混在して結合する

文字どおり, 通常の文字列定数とワイド文字列を混在して結合することができます. ただし, -traditionalを指定するとエラーになります.

リスト4のコンパイル結果と実行結果を次に示します.

```
$ gcc test109.c -o test109
$ ./test109
test=aaa
$ gcc -ansi -pedantic test109.c -o test109
$ ./test109
test=aaa
$ gcc -traditional test109.c -o test109
In file included from
                 /usr/include/features.h:283,
                  from
                  /usr/include/inttypes.h:26,
                  from test109.c:4:
/usr/include/sys/cdefs.h:31: #error
"You need a ISO C conforming compiler to use
 the glibc headers"
$
```

定数の指定方法

● ユニバーサルキャラクタ

従来の仕様では，1バイト文字列または文字定数の定義については明確に定められていましたが，2バイト文字に関しては定められていませんでした．

C99規格では，この記法をユニバーサルキャラクタと呼んでいます．たとえば，

```
const wchar_t str[] = L"¥u4E16¥u754C";
```

のような記法ができるはずなのですが，GCCではまだ対応できていません．unicodeを使用して，日本語，英語以外の言語でプログラミングする場合に便利な機能だと思いますが，通常は必要ないと思います．

● プログラム中で使用できる文字種

従来の仕様では識別名に，マルチバイト文字やユニバーサルキャラクタを使用することは不可能でした．しかし，C99規格ではそれが可能になりましたが，GCCではまだ対応ができていません．

unicodeならともかく，環境に依存したマルチバイト文字を使用した場合，可搬性に大きな問題が発生してしまいます．したがって，実装されても使い道はないように思えます．

● 浮動小数点定数を16進数で表記する

浮動小数点の誤差を減らすために，定数の表記を以下のように変更できます．

```
float  f  =  0x000f.ddddP0;
```

0x000fの部分は整数で16進値，ddddの部分が小数で16進値，最後のゼロは指数です．

GCCでは，拡張機能として導入されていました．-ansiを指定した場合にはエラーになります．

リスト5の実行結果を以下に示します．

```
$ gcc test110.c -o test110
$ ./test110
15.866653
$ gcc -ansi -pedantic test110.c -o test110
test110.c: In function `main':
test110.c:6: floating constant may not be
                                in radix 16
$
```

配列について

● 可変長自動配列

C99規格では，可変長自動配列を使用することができます．連載第7回の拡張機能として説明したものと同一です．

● 長さが0の配列

C99規格では，長さが0の配列を使用することができます．これもまた，GCCの拡張機能としてすでに実装されています．-ansiを指定した場合にはエラーになります．

リスト6のコンパイル結果は，以下のようになります．

```
$ gcc test111.c -o test111
$ gcc -ansi -pedantic test111.c  -o test111
test111.c:11: warning: file does not end in
                                    newline
test111.c:6: warning: ANSI C forbids zero
                       -size array `youso2'
$
```

● 配列要素に記憶修飾子，型修飾子を記述する

C99規格では，配列要素に記憶修飾子や型修飾子を記述することが可能になりましたが，GNU Cはこれに対応していません．コンパイルエラーになります．

リスト7のコンパイル結果を以下に示します．

```
$ gcc test112.c -o test112
```

〔リスト4〕文字列定数とワイド文字列定数を混在して結合する(test109.c)

```
/*
 * 文字列定数とワイド文字列定数を混在して結合する
 */
#include <inttypes.h>
main()
{
    wprintf(L"test=" "aaa" "¥n","¥n");
}
```

〔リスト5〕浮動小数点定数の表記(test110.c)

```
/*
 * 浮動小数点の表記
 */
main()
{
    float  f  =  0x000f.ddddP0;
    printf("%f¥n",f);
}
```

〔リスト6〕長さが0の配列(test111.c)

```
/*
 * 長さが0の配列
 */
typedef struct zerotbl {
  int youso1;
  int youso2[0];
}zerotblstr;
main()
{
    zerotblstr  *str  =  (zerotblstr  *)malloc(sizeof(zerotblstr) + sizeof(int) * 10);
}
```

〔リスト7〕記憶修飾子，型修飾子を記述(test112.c)

```
/*
 * 配列要素に記憶修飾子、型修飾子を記述する
 */
typedef struct zerotbl {
  int youso1;
  int youso2[0];
}zerotblstr;
main()
{
    zerotblstr  *str  =  (zerotblstr  *)malloc(sizeof(zerotblstr) + sizeof(int) * 10);
    test1(int test1_1[static 100]);
    test2(int test2_1[restrict],int test2_2[restrict ]);
    test3(int test3_1[const]);
    test4(int test4_1[volatile]);
}
```

〔リスト8〕C99規格の_Bool型(test113.c)

```
/*
 *_Bool 型
 */
#include <stdbool.h>
main()
{
    _Bool     a;
    printf("%d¥n",sizeof(_Bool));
    printf("%d¥n",sizeof(short));
    printf("%d¥n",sizeof(int));
}
```

〔リスト9〕従来のbool型およびそのサイズ(test114.c)

```
/*
 *_Bool 型
 */
#include <stdbool.h>
main()
{
    printf("bool 型の大きさ =%d¥n",sizeof(bool));
    printf("short 型の大きさ =%d¥n",sizeof(short));
    printf("int 型の大きさ =%d¥n",sizeof(int));
}
```

```
test112.c: In function `main':
test112.c:11: parse error before `int'
test112.c:12: parse error before `int'
test112.c:13: parse error before `int'
test112.c:14: parse error before `int'
$
```

bool型について

言語仕様の中では規定されていなかったのですが，たいていのC言語環境ではboolが定義されています．C99規格では，以下のように規定されました．

- bool型は符号なしの型として扱われる
- bool型は0と1を格納できれば十分なサイズ
- bool型に変換する場合の値は0か1である
- bool型は標準整数型よりもサイズが小さい
- bool型をビットフィールドに定義することができる

そして重要なことは，今まで定義されていたboolとコンフリクトしないように，_Bool型という名称に変更されたことです．ただし，本連載で基準としているGCC2.95では対応できていません．

リスト8と**リスト9**のコンパイルおよび実行結果を以下に示します．

```
$ gcc -S test113.c
test113.c: In function `main':
test113.c:7: `_Bool' undeclared (first use
                             in this function)
test113.c:7: (Each undeclared identifier is
                            reported only once
test113.c:7: for each function it appears
                                          in.)
test113.c:7: parse error before `a'
$ gcc -S test114.c
$
$ gcc  test114.c  -o test114
$ ./test114
bool 型の大きさ= 4
short 型の大きさ= 2
```

```
int型の大きさ= 4
$
```

このように，_Boolではコンパイルが通りません．そして，bool型のサイズも従来と変わりません．

long long int型

従来のlong型を使って，たとえばお金に関する計算をしようとすると，そのままでは10億の単位が最大でした．これでは地方自治体の予算さえ扱えません．しかし，C99規格で規定されたlong long int型を使えば，1000京まで扱えます．これで，トルコリラでもウォンでもペソでも計算できるでしょう．

そのサイズに関しては，ヘッダファイルlimits.hで定義されています．

符号付きの場合は，絶対値9,223,372,036,854,775,807まで保持できます．また，符合なしの場合，その2倍に1を加えた18,446,744,073,709,551,615まで保持できます．

なお，64ビット環境ではlong型とlong long int型は同等です．

もっとも，GNU CやUNIX環境の一部では，以前からlong long int型が定義されていました．これも，それぞれの環境で取り入れていた拡張機能が標準になった例です．

リスト10の実行例を以下に示します．

```
$ gcc test115.c -o test115
$ ./test115
longlong int型の大きさ= 8
big_intの値= 9223372036854775807
$
```

ソースを見るとわかるとおり，定数として扱う場合のサフィックスはLL，llです．符号付きはUが付いてULL，ullです．

また，以下のように-ansiオプション付きでコンパイルすると，警告が出ます．

```
$ gcc -ansi -pedantic test115.c -o test115
test115.c:10: warning: file does not end in
                                        newline
test115.c: In function `main':
test115.c:7: warning: ANSI C does not
                              support `long long'
test115.c:7: warning: ANSI C forbids long
                              long integer constants
test115.c:8: warning: ANSI C does not
                              support `long long'
$
```

これについては，連載第3回で触れています．

整数除算に関する規定

従来のC言語の規格では，両方のオペランドが正の数の場合，商は小数点以下を捨てた数になることになっています．たとえば，1/2の場合，正しい値は0.5ですが，小数点以下を捨てるので商は0，余りが1になります．

どちらか一方の数が負の場合の計算結果は，環境に依存していましたが，C99規格で規定されることになりました．それは，割り算の結果はすべて0方向に切り捨てるということです．

リスト11のソースの実行結果を以下に示します．

```
$ gcc test116.c -o test116
$ ./test116
-1/2 =0
-1.0f/2.0f =-0.500000
-1/2.0f =-0.500000
-1.0f/2 =-0.500000
$
```

上記のように，オペランドの両方が整数の場合は0方向に切り捨てられます．

どちらか，または一方が浮動小数点型の場合には，結果は切り捨てられません．

複素数型

_Complex型は，C99規格で決定される以前から通常の環境で拡張機能として利用されていました．

通常の環境では，構造体で実数部と虚数部を定義して実装しています．

GNU Cでは実数部をレジスタに置き，虚数部をスタック上に置くことも，またその逆も可能ですが，その状態を管理できるデバッガがありません．

〔**リスト10**〕**long long int型**（test115.c）

```
/*
 *long long int型
 */
#include <stdio.h>
main()
{
    long long int    big_int = 9223372036854775807LL;
    printf("longlong int型の大きさ =%d¥n",sizeof(long long int));
    printf("big_intの値 =%lld¥n",big_int);
}
```

〔**リスト11**〕**整数除算に関する規定**（test116.c）

```
/*
 * 整数の除算
 */
#include <stdio.h>
main()
{
    printf("-1/2 =%d¥n",-1/2 );
    printf("-1.0f/2.0f =%f¥n",-1.0f/2.0f );
    printf("-1/2.0f =%f¥n",-1/2.0f );
    printf("-1.0f/2 =%f¥n",-1.0f/2 );
}
```

複素数の自動変数を非連続的な形で割り当てることができますが，その場合，あたかもそれが二つの別々の非複素数変数であるかのように記述するので，デバッグの際に注意してください．

虚数定数にはサフィックスiが付きます．よって，float _Complexは，10.25f + 25.65iなどと表記することが可能です．

リスト12の実行結果を以下に，生成されたアセンブラソースを**リスト13**に示します．

```
$ gcc -S test117.c
$ gcc test117.c -o test117
$ ./test117
50.235001
15.253000
2
1
$
```

複素数における虚数部の値を示すための型である_Imaginary型については，GNU C2.95ではまだサポートされていません．

リスト12のソースのように，

```
aai = __imag__ ai;
bbi = __real__ ai;
```

のような使い方ができるので，不便ではないと思います．

次回でバージョンごとのC99規格の実装を明確にする予定ですが，GCC3.3でも実装されていないようです．

〔リスト12〕簡単な複素数の計算(test117.c)

```
/*
 *_Complex
 */
#include <stdio.h>
#include <complex.h>
main()
{
    float  _Complex    a;
    int    _Complex    ai;
    double _Complex    b;
    long   double _Complex    c;
    float  aa;
    float  bb;
    int    aai;
    int  bbi;
    a    =    15.253f   +    (float _Complex)50.235i;
    aa   =    __imag__ a;
    bb   =    __real__ a;
    printf("%f¥n",aa);
    printf("%f¥n",bb);
    ai   =    1    +    (int _Complex)2i;
    aai  =    __imag__ ai;
    bbi  =    __real__ ai;
    printf("%d¥n",aai);
    printf("%d¥n",bbi);
}
```

暗黙の関数宣言と変数宣言

● 暗黙の関数宣言

従来の標準規格では，関数のプロトタイプを宣言しなくても使用することができました．プロトタイプについては連載第8回でも触れていますが，重要な問題です．エラーにならなくてもプロトタイプ宣言すべきだと思います．

● 暗黙の変数宣言

従来の標準規格では，

```
const x = 10;
```

のような使い方ができました．

この場合，xはint型になりますが，C99規格ではこのような使用法はできないことになりました．しかし，GNU Cではまだできています．

以下に，**リスト14**のコンパイルと実行結果を示します．

```
$ gcc test118.c -o test118
$ ./test118
1
$ gcc -ansi -pedantic test118.c -o test118
$ ./test118
```

〔リスト13〕生成されたアセンブラソース(test117.s)

```
    .file    "test117.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
    .align 4
.LC0:
    .long 0x41740c4a
    .long 0x4248f0a4
.LC1:
    .string  "%f¥n"
    .align 4
.LC2:
    .long 1
    .long 2
.LC3:
    .string  "%d¥n"
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $104,%esp
    flds .LC0
    fstps -8(%ebp)
    flds .LC0+4
    fstps -4(%ebp)
    flds -4(%ebp)
    fstps -68(%ebp)
    flds -8(%ebp)
    fstps -72(%ebp)
    addl $-4,%esp
    flds -68(%ebp)
    subl $8,%esp
    fstpl (%esp)
    pushl $.LC1
    call printf
    addl $16,%esp
    addl $-4,%esp
    flds -72(%ebp)
    subl $8,%esp
    fstpl (%esp)
    pushl $.LC1
    call printf
    addl $16,%esp
    movl .LC2,%eax
    movl %eax,-16(%ebp)
    movl .LC2+4,%eax
    movl %eax,-12(%ebp)
    movl -12(%ebp),%eax
    movl %eax,-76(%ebp)
    movl -16(%ebp),%eax
    movl %eax,-80(%ebp)
    addl $-8,%esp
    movl -76(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
    addl $-8,%esp
    movl -80(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト 14〕暗黙の変数宣言(test118.c)

```
/*
 * 暗黙の型
 */
#include <complex.h>
main()
{
    const    x    =    1;
    printf("%d¥n",x);
}
```

〔リスト 15〕定義済みマクロ __func__ を使ったデバッグ(test119.c)

```
/*
 * デバッグに役立つ定義済みマクロ
 */
#define dbg(...)  (printf("%s %u @%s:",__FILE__,__LINE__,__func__),printf(" "__VA_ARGS__))
#include <complex.h>
void foo1(int i);
void foo2(int i);
void foo3(int i);
void foo4(int i);
main()
{
    int  x    =    1;
    foo1(x++);
    foo2(x++);
    foo3(x++);
    foo4(x);
}
void foo1(int i)
{
        dbg("i=%d¥n", i);
}
void foo2(int i)
{
        dbg("i=%d¥n", i);
}
void foo3(int i)
{
        dbg("i=%d¥n", i);
}
void foo4(int i)
{
        dbg("i=%d¥n", i);
}
```

〔リスト 16〕生成されたアセンブラソース(test119.s)

```
    .file    "test119.c"
    .version "01.01"
gcc2_compiled.:
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl $1,-4(%ebp)
    addl $-12,%esp
    movl -4(%ebp),%eax
    pushl %eax
    incl -4(%ebp)
    call foo1
    addl $16,%esp
    addl $-12,%esp
    movl -4(%ebp),%eax
    pushl %eax
    incl -4(%ebp)
    call foo2
    addl $16,%esp
    addl $-12,%esp
    movl -4(%ebp),%eax
    pushl %eax
    incl -4(%ebp)
    call foo3
    addl $16,%esp
    addl $-12,%esp
    movl -4(%ebp),%eax
    pushl %eax
    call foo4
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
.section .rodata
.LC0:
    .string   "foo1"
.LC1:
    .string   "test119.c"
.LC2:
    .string   "%s %u @%s:"
.LC3:
    .string   " i=%d¥n"
.text
    .align 4
.globl foo1
    .type     foo1,@function
foo1:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    pushl $.LC0
    pushl $20
    pushl $.LC1
    pushl $.LC2
    call printf
    addl $16,%esp
    addl $-8,%esp
    movl 8(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
.L3:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     foo1,.Lfe2-foo1
.section .rodata
.LC4:
    .string   "foo2"
.text
    .align 4
.globl foo2
    .type     foo2,@function
foo2:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    pushl $.LC4
    pushl $24
    pushl $.LC1
    pushl $.LC2
    call printf
    addl $16,%esp
    addl $-8,%esp
    movl 8(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
.L4:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe3:
    .size     foo2,.Lfe3-foo2
.section .rodata
.LC5:
    .string   "foo3"
.text
    .align 4
.globl foo3
    .type     foo3,@function
foo3:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    pushl $.LC5
    pushl $28
    pushl $.LC1
    pushl $.LC2
    call printf
    addl $16,%esp
    addl $-8,%esp
    movl 8(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
.L5:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe4:
    .size     foo3,.Lfe4-foo3
.section .rodata
.LC6:
    .string   "foo4"
.text
    .align 4
.globl foo4
    .type     foo4,@function
foo4:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    pushl $.LC6
    pushl $32
    pushl $.LC1
    pushl $.LC2
    call printf
    addl $16,%esp
    addl $-8,%esp
    movl 8(%ebp),%eax
    pushl %eax
    pushl $.LC3
    call printf
    addl $16,%esp
.L6:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe5:
    .size     foo4,.Lfe5-foo4
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト17〕列挙型(test120.c)

```
/*
 *列挙型
 */
#include <stdio.h>
    enum cafe
    {
         grande,
         tall,
         small,
    };
main()
{
    enum cafe JAVA;
    JAVA =    tall;
    printf("%d¥n",JAVA);
}
```

〔リスト18〕生成されたアセンブラソース(test120.s)

```
    .file    "test120.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
.LC0:
    .string  "%d¥n"
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-8,%esp
    pushl $0
    pushl $.LC0
    call printf
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

```
1
$
```

どちらの方法でもエラーになりません.

定義済みマクロ__func__と可変長引き数マクロ

リスト15のようなソースなら絶対に追えますが，初心者に作られたスパゲッティプログラムを追うのは嫌なものです．そんな場合，関数内に定義済みマクロ__func__を表示するprintfを入れると，非常にわかりやすくなって，スパゲッティプログラムをデバッグするのが楽しくなります(?).

この機能は，GNU Cの拡張機能として以前から使われていましたが，C99規格にも取り入れられました.

以下に，**リスト15**のコンパイルおよび実行結果を示します．-ansiオプションを使用するとエラーになります.

```
$ gcc test119.c -o test119
$ ./test119
test119.c 20 @foo1: i=1
test119.c 24 @foo2: i=2
test119.c 28 @foo3: i=3
test119.c 32 @foo4: i=4
$

$ gcc -ansi -pedantic test119.c -o test119
test119.c:4: warning: invalid character in
                          macro parameter name
test119.c:4: badly punctuated parameter
                               list in `#define'
```

リスト16のアセンブラソースを見てわかるとおり，各関数の最初でマクロの処理をしています.

列挙型について

従来の規格では，ソース例のようにenumの項目に余分なカンマがあると，ワーニングを出しましたが，C99規格ではそれを許しています．よって，-ansiオプションを使用するとワーニングになります.

リスト17のコンパイルと実行結果を下記に示します．また，生成されたアセンブラを**リスト18**に示します.

```
$ gcc -ansi -pedantic test120.c -o test120
test120.c:10: warning: comma at end of
                               enumerator list
$ gcc test120.c -o test120
$ ./test120
1
$
```

inline関数定義

基本的にC++言語の仕様と同じものです．従来の規格で，高速な関数呼び出しを行う場合にはマクロを使用していました.

しかし，型検査ができなかったり，副作用を起こすといった問題がありました．このC99規格はその問題を解決させるものです.

ただし，GNU Cではまだ対応できていません.

restrictポインタ

最適化と効率の良いコードを生成するためには，不可欠な機能です．C99規格の目玉の一つでしょう.

簡単にいえば，「そのポインタ変数は同一の番地を絶対に指していない」とコンパイラに教えるためにあります．たとえば，その関数内で二つのポインタ変数が別の番地を指して，どちらもメモリ内部の値を更新していると仮定します．一つ目のポインタ変数が00030210番地を更新した後，二つ目のポインタがある番地を更新する際に00030210番地の内容が必要なとき，00030210番地を更新した値をそのまま使ったほうが効率がよいはずです.

問題は，二つのポインタが同じ番地を指していたら，お互い

〔リスト19〕変数宣言の位置(1)
(test121.c)

```
/*
 * 変数の宣言場所(1)
 */
#include <stdio.h>
main()
{
    char buf1[10240];
    memset(&buf1,0,sizeof(buf1));
    int i   =   1;
    for  (i=0;i<sizeof(buf1);i++)
    {
        buf1[i]   =   i;
    }
}
```

〔リスト20〕変数宣言の位置(2)
(test122.c)

```
/*
 * 変数の宣言場所(2)
 */
#include <stdio.h>
main()
{
    int  i   =   1;
    char buf1[10240];
    memset(&buf1,0,sizeof(buf1));
    for  (int i=0;i<sizeof(buf1);i++)
    {
        buf1[i]   =   i;
    }
}
```

〔リスト21〕コンパウンドリテラル(test123.c)

```
/*
 *Compound Literal
 */
#include <stdio.h>
typedef struct { int x, y; } XY_struct ;
int test(const XY_struct *p);
main()
{
    int  res;
    res  =   test(&(XY_struct){100, 200});
    printf("%d¥n",res);
    res  =   test(&(XY_struct){500, 235});
    printf("%d¥n",res);
}
int test(const XY_struct *p)
{
    return (p->x * p->y);
}
```

に干渉し，その値が不定かもしれないという点です．

そんなとき，コンパイラに「そのポインタ変数は同一の番地を絶対に指していない」と教えることで，効率のよいコードが作成できます．

これはGNU Cでも対応しています．

変数宣言の位置

C++では，変数の宣言を使用する行で行っても問題はありません．

しかし，従来の規格ではグローバル変数は関数を定義する直前までに，また関数内部で使用する局所変数は，そのコードのブロックの先頭で行わなければなりませんでした．

C99規格では，変数の宣言の位置に関してはC++風になりました．

GNU Cではまだ未対応です．

リスト19のようなコードの場合，コンパイル時に以下のようなことになります．

```
$ gcc -S  test121.c
test121.c: In function `main':
test121.c:9: parse error before `int'
test121.c:10: `i' undeclared (first use in
                                this function)
test121.c:10: (Each undeclared identifier
                         is reported only once
test121.c:10: for each function it appears
                                           in.)
$
```

また，**リスト20**のようなコードでは，次のような結果になります．

```
$ gcc -S  test122.c
test122.c: In function `main':
test122.c:10: parse error before `int'
test122.c:10: parse error before `)'
$
```

配列の初期化について

C99規格では配列を初期化する際に，配列の要素やメンバ名を指定することで，わかりやすく初期化を行えます．連載第7回，「拡張機能」の中で配列の初期化について例を挙げて解説してあるので参考にしてください．

GNU Cでは，拡張機能として実装されていました．非常に可読性が高まるので，使ってみてください．

コンパウンドリテラル (Compound Literal)

構造体を引き数として関数に渡したいときに，構造体を確保して，変数をセットして……としなくてもできるようになりました．

もちろん，すべてこのようにしたら可読性に問題が出ますが，デバッグなどの際には非常に便利だと思います．

GNU Cも対応しています．

コンパウンドリテラルを使った例を**リスト21**に，その結果を下記に示します．また，生成されたアセンブラソースを**リスト22**に示します．

```
$ gcc test123.c -o test123
$ ./test123
20000
117500
$
```

また，コンパウンドリテラルを使用せず，従来の方法で行った例を**リスト23**に示します．実行結果を以下に，生成されたアセンブラソースを**リスト24**に示します．

```
$ gcc test124.c -o test124
$ ./test124
20000
117500
```

〔リスト22〕生成されたアセンブラソース(test123.s)

```
    .file    "test123.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
    .align 4
.LC0:
    .long 100
    .long 200
.LC1:
    .string  "%d¥n"
    .align 4
.LC2:
    .long 500
    .long 235
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    addl $-12,%esp
    pushl $.LC0
    call test
    addl $16,%esp
    movl %eax,%eax
    movl %eax,-4(%ebp)
    addl $-8,%esp
    movl -4(%ebp),%eax
    pushl %eax
    pushl $.LC1
    call printf
    addl $16,%esp
    addl $-12,%esp
    pushl $.LC2
    call test
    addl $16,%esp
    movl %eax,%eax
    movl %eax,-4(%ebp)
    addl $-8,%esp
    movl -4(%ebp),%eax
    pushl %eax
    pushl $.LC1
    call printf
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .align 4
.globl test
    .type     test,@function
test:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 8(%ebp),%ecx
    movl (%eax),%edx
    imull 4(%ecx),%edx
    movl %edx,%eax
    jmp .L3
    .p2align 4,,7
.L3:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     test,.Lfe2-test
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

〔リスト23〕同じことを通常の方法で(test124.c)

```
/*
 *Compound Literal
 */
#include <stdio.h>
typedef struct { int x, y; } XY_struct ;
int test(const    XY_struct *p);
main()
{
    int  res;
    XY_struct var1;
    var1.x    =    100;
    var1.y    =    200;
    res  =    test(&var1);
    printf("%d¥n",res);
    var1.x    =    500;
    var1.y    =    235;
    res  =    test(&var1);
    printf("%d¥n",res);
}
int test(const  XY_struct  *p)
{
    return  (p->x  *  p->y);
}
```

〔リスト24〕生成されたアセンブラソース(test124.s)

```
    .file    "test124.c"
    .version "01.01"
gcc2_compiled.:
.section .rodata
.LC0:
    .string  "%d¥n"
.text
    .align 4
.globl main
    .type     main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $24,%esp
    movl $100,-12(%ebp)
    movl $200,-8(%ebp)
    addl $-12,%esp
    leal -12(%ebp),%eax
    pushl %eax
    call test
    addl $16,%esp
    movl %eax,%eax
    movl %eax,-4(%ebp)
    addl $-8,%esp
    movl -4(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call printf
    addl $16,%esp
    movl $500,-12(%ebp)
    movl $235,-8(%ebp)
    addl $-12,%esp
    leal -12(%ebp),%eax
    pushl %eax
    call test
    addl $16,%esp
    movl %eax,%eax
    movl %eax,-4(%ebp)
    addl $-8,%esp
    movl -4(%ebp),%eax
    pushl %eax
    pushl $.LC0
    call printf
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe1:
    .size     main,.Lfe1-main
    .align 4
.globl test
    .type     test,@function
test:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax
    movl 8(%ebp),%ecx
    movl (%eax),%edx
    imull 4(%ecx),%edx
    movl %edx,%eax
    jmp .L3
    .p2align 4,,7
.L3:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfe2:
    .size     test,.Lfe2-test
    .ident    "GCC: (GNU) 2.95.3 20010315 (release)"
```

$

コンパウンドリテラルを使用したソースで生成したアセンブラでは，引き数が定数になっています．従来の方式で生成したアセンブラでは，スタックに転送しています．

引き数が定数になっていたほうが，理屈では速いような気がします．

isblank

isblankは，C99規格で新しく導入されました．ctype.hヘッダで宣言されています．今までにもisspaceがありましたが，isblankは純粋に空白をチェックします．

〔リスト 25〕**isblank の簡単な試験**(test125.c)

```
/*
 *isblank
 */
#include <stdio.h>
main()
{
    char buf[10];
    memset(&buf,' ',sizeof(buf));
    if   (isblank(buf[0])   !=   0    )    printf("' '=Blank¥n");
    if   (isblank(buf[0])   !=   0    )    printf("' '=space¥n");
    buf[1]    =    (char)"a";
    if   (isblank(buf[1])   !=   0    )    printf("a=Blank¥n");
    if   (isblank(buf[1])   !=   0    )    printf("a=space¥n");
    buf[2]    =    '¥n';
    if   (isblank(buf[2])   !=   0    )    printf("¥¥n=Blank¥n");
    if   (isspace(buf[2])   !=   0    )    printf("¥¥n=space¥n");
    buf[3]    =    '¥r';
    if   (isblank(buf[3])   !=   0    )    printf("¥¥r=Blank¥n");
    if   (isspace(buf[3])   !=   0    )    printf("¥¥r=space¥n");
    buf[4]    =    '¥t';
    if   (isblank(buf[4])   !=   0    )    printf("¥¥t=Blank¥n");
    if   (isspace(buf[4])   !=   0    )    printf("¥¥t=space¥n");
    buf[5]    =    '¥v';
    if   (isblank(buf[5])   !=   0    )    printf("¥¥v=Blank¥n");
    if   (isspace(buf[5])   !=   0    )    printf("¥¥v=space¥n");
    buf[6]    =    '¥a';
    if   (isblank(buf[6])   !=   0    )    printf("¥¥a=Blank¥n");
    if   (isspace(buf[6])   !=   0    )    printf("¥¥a=space¥n");
    buf[7]    =    '¥b';
    if   (isblank(buf[7])   !=   0    )    printf("¥¥b=Blank¥n");
    if   (isspace(buf[7])   !=   0    )    printf("¥¥b=space¥n");
    buf[8]    =    '¥f';
    if   (isblank(buf[8])   !=   0    )    printf("¥¥f=Blank¥n");
    if   (isspace(buf[8])   !=   0    )    printf("¥¥f=space¥n");
}
```

isspace は復帰コード，改行コードも空白と認識してしまいます．

isblank は，¥t(水平タブ)も空白と認識します．¥v(垂直タブ)は空白と認識しません．

リスト 25 に，isblank のテストを行うコードを，そして実行結果を以下に示します．

```
$ gcc   test125.c -o test125
$ ./test125
' '=Blank
' '=space
¥n=space
¥r=space
¥t=Blank
¥t=space
¥v=space
¥f=space
$
```

結果をまとめると，**表 1** のようになります．用途によって使い分けることができます．もちろん，GNU C でも使うことができます．

これ以降の関数やマクロに関しては，誌面の都合で検証および詳細な説明は次回に行います．ここでは簡単な説明にとどめておきます．

〔表 1〕**isspace と isblank の比較**

	isspace	isblank
空白(0x20)	真	真
¥n 復帰改行	真	偽
¥r 復帰	真	偽
¥f 改ページ	真	偽
¥t 水平タブ	真	真
¥v 垂直タブ	真	偽
¥a ベル	偽	偽
¥b 後退	偽	偽

関数やマクロの概略

● マクロ DECIMAL_DIG

float.h で宣言されているはずですが，GNU C2.95 では存在しません．C99 規格で新しく導入されました．

浮動小数点型で表現できる最大の 10 進桁数が定義されているはずです．

● マクロ FLT_EVAL_METHOD

float.h で宣言されているはずですが，GNU C2.95 では存在しません．C99 規格で新しく導入されました．

浮動小数点演算を行うときの範囲と精度を示す値が定義されているはずです．

● math.h の新しいマクロや関数

C99 規格で拡張または追加された math.h 関連のマクロや関数について GNU C2.95 では，まだ未対応です．

公式ページの情報によるとライブラリに問題があるようです．

● マクロ va_copy

va_copy は C99 規格で新たに導入されたマクロで，stdarg.h ヘッダで定義されています．GNU C2.95 でも対応しています．可変長引き数リストのコピーを行うために使用します．

他の環境ではそのまま“va_copy”かもしれませんが，GNU C では，__va_copy として定義されています．

以下の一連の流れをマクロ化したものです．

1) va_start で可変長引き数への参照のためのオブジェクトを初期化
2) va_arg で型を指定して可変長引き数を取り出す
3) va_end を使用してオブジェクトを破棄

つまり，va_start でメモリを割り当て，パラメータを格納し，次のパラメータがどれかを指し示すようにします．そして va_arg でリストを順番にたどり，va_end で割り当てたメモリを開放します．

● stdio.h に追加されたいくつかの関数

● snprintf

これは，sprintf に出力文字数を指定できるようにしたものです．宣言は次のとおりです．

```
int snprintf(char *str, size_t size,
                         const char *format, ...);
```

● vsnprintf

実際はstdarg.hに追加されたものですが，ここで説明します．

snprintfの可変数引き数の代わりにva_listを用いて呼び出しを行うものです．これらのva_listを使う関数では，va_endマクロは呼び出されません．メモリの解放は使う側の責任です．

宣言は以下のとおりです．

```
int vsnprintf(char *str, size_t size,
          const char *format, va_list arg);
```

● vscanf, vsscanf, vfscanf

va_listを引き数に持つscanf関連の関数です．

```
int vscanf(const char *format, va_list arg);
int vsscanf(const char *str,
          const char *format, va_list arg);
int vfscanf(FILE *stream,
          const char *format, va_list arg);
```

● stdlibに追加されたいくつかの関数

● _Exit

signalや終了関数を呼び出さずに，プログラムを終了コードstatusですぐに終了させます．_exit関数と等価です．

```
void _Exit(int status);
```

● strtof, strtold関数

strtofはfloatに，strtoldはlong doubleに対応しています．

float型(strtof)，またはlong double型(strtold)を文字列に変換します．

```
float strtof(const char *nptr,
                          char **endptr);
long double strtold(const char *nptr,
                          char **endptr);
```

● strtoll, strtoull関数

見てわかるようにlong long int型を文字列に変換する関数です．

```
strtoll(const char *nptr, char **endptr,
                              int base);
strtoull(const char *nptr, char **endptr,
                              int base);
```

● atoll, llabs, lldiv関数

long long int型関係です．

atollは文字列をlong long int型の値として変換します．

```
long long atoll(const char *nptr);
```

llabsはlong long int型整数の絶対値を計算します．

```
long long int llabs(long long int j);
```

lldivはlong int型の割算の商と余りを計算するものです．

```
ldiv_t lldiv(long long int numer,
                      long long int denom);
```

● wchar.hに追加されたいくつかの関数

● vwscanf, vswscanf, vfwscanf

va_listを引き数にもつscanf関係の関数です．

vsscanfなどがワイド文字列に対応しているものです．

```
int vwscanf(const wchar_t *  format,
                              va_list arg);
int vswscanf(const wchar_t *  s,
        const wchar_t *  format, va_list arg);
int vfwscanf(FILE *fp,
        const wchar_t *format, va_list arg);
```

● wcstof, wcstold

ワイド文字列対応で浮動小数点文字列をfloatやlong doubleに変換するものです．

```
float wcstof(const wchar_t *restrict p,
                          wchar_t ** endp);
long double wcstold(const wchar_t
              *restrict p, wchar_t ** endp);
```

● wcstoll, wcstoull

ワイド文字列対応で文字列をlong long int, unsigned long long intに変換するものです．

```
long long int wcstoll(const wchar_t * p,
                wchar_t ** endp, int base);
unsigned long long int wcstoull
        (const wchar_t * p, wchar_t ** endp,
                              int base);
```

● wctype.hに追加された関数

● iswblank

isblankのワイド文字列対応版です．結果は**表1**と同じになるはずです．

*　　　　　*

C99関連の話題はつきませんが，ここで一区切りとします．次回は，次の項目に対して説明と検証を行います．

● 追加された関数・マクロについて
● 新規に追加された標準ライブラリについて
● C99規格対応に関するGNU Cのバージョンごとの進捗

きし・てつお

● 癒しの音楽

ここのところ，月に一度は秋葉原に行くのを常としている．丸一日，秋葉原界隈をブラブラし，初物や特売品を探し回って自己満足しているのが楽しい．最後に，大好きなラーメンを食べて帰ってくるのだが，じつはそれが最大の楽しみであったりする．

ある日，いつものように秋葉原の裏手を歩いていると，どこからともなく，昔懐かしい曲が耳に飛び込んできた．思わず足を止め，そのゆっくりしたテンポの曲に聴き入ってしまった．しかし，どうしても曲名を思い出せない．行き慣れた店に入ると，音の方向に自然と足が向かった．たどり着いたところには，30年前の（懐かしい音楽がそう見えさせたのかもしれない）オーディオコンポを髣髴(ほうふつ)とさせる立派なスピーカシステムを左右に何台も従えたパソコンがあった．最近のパソコンはいい音が出るものだ，などと感心しつつも，右手首でリズムに合わせている自分にハッと気がついた．もしかしたら，間抜けなオジサンだったかもしれない．周りの視線がちょっと気になった．

何の曲か思い出せなかったが，ビートルズの初期のナンバーであることは間違いない．マイナな曲なので，聞いたのはおそらく30年ぶりのはずだ．この曲を聞いていると，当時のことが鮮明に思い起こされ，その場でしばし感慨にふけっていた．だから手首が動いていても気がつかなかったのだろう．音楽は，さまざまな過去を思い起こさせてくれ，一時的に心を癒してくれることが多い．とくに長い間，聞かなかった曲であればあるほど，その懐かしい思い出は鮮烈にフラッシュバックする．

もちろん若い世代の人にも，当然懐かしい曲が存在するだろう．思うに，懐かしい曲は総じてスローテンポが多いような気がする．今の世の中，スピードが速い．体内時計はハイテンポで進んでいる．こんなとき，時計を戻してゆっくり動かしてくれる曲は，癒しをかもし出してくれるのかもしれない．

休む暇もない速い時間の流れの中で泳がされてきた若い人たちが，この種の歌を聴いて癒されているのはわかる気がする．一方で，その速い時代の流れを自ら自分に果たして突っ走ってきた40，50代の人々――ここでは中高年と呼ぼう――が癒されるのは，また違った意味があるかもしれない．

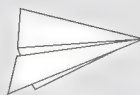
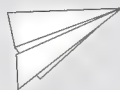
● 高度成長を思い起こせば

中高年では若い部類に属するかもしれない，筆者の年代の話を聞いていただけるだろうか．高度成長の始まりに小中学校教育を受けた．当時の日本は，東京オリンピックや首都高速，新幹線に象徴される，欧米諸国に追いつけ追い越せの時代で，GNPが驚異的なスピードで伸びていた．

学校では，先生がしつけと，一所懸命頑張って競争することを教えた．先生の体罰が怖かったからかもしれない．せっかくの夏休みや冬休みも，信じられないほどの大量の宿題に対しても，決して文句をいえず驚きもしなかった．休み中は少し憂鬱に過ごしつつも，終わりに近づくと，何日も何日も徹夜で頑張って，登校日までには宿題を片付けた．おかげで集中力が磨き上げられ，どれくらいの時間でどれくらいの量をこなせるかという感覚も身に付けることができた．

そうして，会社に入ったら，先輩たちはバリバリ働く団塊の世代で，例外なく鍛えられた企業戦士だった．先輩たちの指示に従順にしたがい，文句一ついわず組織一丸となって高い目標に向かい，わき目も振らず邁進していった．そんな具合だったから，会社を休むときも後めたかったし，有給休暇のほとんどを捨てるのも例年のことだった．子供が生まれても家に戻らずに，職場で自分の責務をまっとうしようとする先輩や友人たちを見て当然だと思い，自分もそうしていた．

その一方で，開発環境は劣悪だった．端末は順番待ちで，一人一台の端末なんて夢のような話だった．だから，ソースは帳票上でデバッグを行った．もちろん，端末を見ながらやるより，そのほうが効率的だと教わったからであり，確かにそうだった．コンパイルするのに数時間かかるのが常識だったから，徹夜で作業してプログラムの修正が終わり，朝方コンパイルをかけ始めると，そのまま寝た．始業のベルが鳴るとコンパイルが終わっていて，そして出ているエラーを帳票に写すのが常だった．

オフィスビルは，夜になると空調も切れて蒸し暑い．そんな劣悪な環境でも，文句一ついわず，スーツを脱ぎ捨ててランニング姿になり，ひたすら働いた．会社を出るのは午前0時をまわり，遅いときは始発だった．それでも朝9時にはきちんと出勤した．遅刻にはとくに厳しかったからだ．もちろん土日の出勤は普通のことだった．TVや新聞を見る機会もあまりなかったため，三原山が大噴火していようと知りもせず，世間とは隔絶していた．たまにTVを見ると，過労死問題が報じられていること

があった．しかし，どのケースを見ても我々ソフトウェア業界の劣悪な労働環境に比べたらおよびもしないと，一種の優越感さえ感じたほどである．

こうして，休みなしにずっと走ってきた．それはそれでよかったかもしれない．納期があまりにもタイトで間に合いそうもなかったり，未修得の技術で完成できなかったりするかもしれないといった状況で，強迫観念を感じながらも，最後には必ず完成させていった．そんなことを繰り返しているうちに，難易度の高い設計や開発を責任もって受けられる技術力，そしてなによりも度胸が身についたように思う．

● やりたくても出来なかった

先日，あるパソコンの試験を受けた．緊張した面持ちのなかに，妙に不釣合いな男が隣にいた．この種の試験を受ける年齢層としては，ちょっと高めだ．筆者のちょっとした話に気楽に乗ってきた彼とは，同世代ということもありけっこううまが合い，試験終了後，昼食を一緒に食べることになった．そこで彼の話を聞いた筆者は，食事が喉を通らなくなってしまった．

彼は有名私立大の院卒で46歳だ．ソフトウェア開発会社で研究開発をしており，大学入試を控えた子供がいるらしい．入社以来，寝る時間も惜しんで研究や仕事に没頭し，過去に会社から2度ほど表彰を受けたこともあるそうだ．数年前，現場からR&D部門に移り，会社に多くの予算をもらってインターネット関連の研究開発を続けてきたそうだが，当初計画したとおりの結果が得られなかった．そんなときに会社が45～55歳を対象にしたリストラを発表し，その対象者として今年の3月に会社を辞めることが決まった．

突然のことだったが，退職後は個人タクシーの運転手をすることに決めたという．不況でもっとも増える業種はタクシーだといわれている．そして，不況時にはタクシーの乗客も減る．だいたい技術者が接客業をやろうというのだから，たいへんな職種を選んだに違いない．しかし，そんなことは重々承知らしい．夜遅いことが平気だという理由で，タクシーの運転手を選ぶまでにはそう時間がかからなかったという．

そうと決まったら，これを機会に仕事でやってきた成果を確かめよう．いままで仕事で忙しくて試験勉強などできなかったので，退職するまでの間に勉強し，いろいろなパソコン関係の

試験を受験することにしたという．

彼と別れ際に，なぜ会社からの話を断らなかったのかと聞いてみた．彼の答は，「会社もつらい状況だから，断ることなどできなかった．それに自分はまだ若い．これから第二の人生を歩みたい」ということだった．

今まで会社のために一所懸命働いてきたのに，業績が悪くなると辞めさせられながらも，文句もいわずに会社を擁護し，自分は前向きに頑張ろうとしている．もう少し時間をかけ，ハローワークなどに通って仕事を探していこうとはせず，次の仕事を決めてしまい，少しでも空いた時間には試験勉強をしている．そんな姿に筆者は心を打たれたし，一方で若干だが中高年の哀れを感じざるを得ない．

彼の選択が誤りであるかどうかはわからない．しかし，突然彼を襲った不幸に対し，上手にスローテンポで生きていく生き方を忘れているような気がしてならない．そんな機会は，30年，40年ぶりに違いないと思うのはひがみなのだろうか．

あさひ・しょうすけ　テクニカルライター
イラスト　森　祐子

Engineering Life in

シリコンバレーに夫婦で出向(第二部)

■今回のゲストのプロフィール
鈴木友子(すずき・ともこ):1992年,愛知教育大学総合科学課程国際文化コース英米文化選修卒業.NECマイコンテクノロジーに入社後,社内でのOn the Job Trainingを受け,組み込みマイコン用ソフトウェアツールのサポート業務に就く.8～64ビットマイコン向け開発ツールの関連業務を8年間行った後,2001年10月にNEC Electronics America, Inc.にProduct Marketing Engineerとして出向し,現在に至る.趣味は油絵,スキー,キーボード.最近買ったDVDは「007」7本セット,最近買ったお気に入りの本は『気がつくと机がぐちゃぐちゃになっているあなたへ』.
鈴木　敦(すずき・あつし):1988年,幾徳工業大学(現神奈川工科大学)卒.同年NECマイコンテクノロジー入社.入社以降十年以上にわたり,NEC独自アーキテクチャの32ビットCPU向け基本ソフトの開発に従事.2001年に米国出向となり,業務内容も開発からサポートへと変わり現在に至る.趣味はドライブ,スキー,読書.日本ではバイクにも乗っていたが,こちらではいまだに免許も取得していない.

前回まで:夫婦の仕事の条件が整いシリコンバレーに出向した背景,そしてシリコンバレーの第一印象についてうかがった.

☆多くのギャップを感じるシリコンバレー

友子 カルチャーショックというか,生活のいろいろなシーンでギャップを感じることが多いですね.そのなかでも,非常に対称的な人々が多いかな?と感じることがよくあります.

たとえば,健康に気を使っている人が多くいます.そのためフィットネスジムが24時間営業していたり,ありとあらゆるサプリメントが売られています.テレビ販売番組でもフィットネス系のグッズが多いですし…….しかし,その一方で太っている人もいます.不健康そうな物を沢山食べているとか,フィットネス系グッズもずぼらな系統が多いなどです.健康オタクな人はとことん凄いし,エンジニアでも筋肉質で凄い人がいます.

トニー あはは!　私がその類です.ほぼ毎日ジムに行っているし,サプリメントなどもけっこう飲んでますよ.私の通っているジムはエンジニアが多く,ヘロヘロになるまで運動してシャワーを浴びてまた仕事に戻る人もいます.メリハリをつけているつもりでしょうか?!

敦 それにレストランなどで食べ物が残っていると,「お持ち帰りにしますか?」と聞かれますよね.

トニー どんなレストランでもありますよね.高級レストランだと綺麗にアルミホイルで白鳥とかバスケットとかの形にしてくれます.

敦 そういう点では無駄遣いがなく良い習慣だと思います.しかし逆に,やたらに紙を使うとも感じています.オフィスではプリントアウトを大量に行うし,休憩室で紙のナプキンなどを大量に消費しています.スーパーでも紙の袋かポリ袋か聞いてくるのはエコロジー重視で素晴らしいと思いますが,一貫していないことに違和感を感じます.

トニー 共働きの夫婦で子供がいる家庭では,平日とか普通の日は食事に紙コップ,紙プレートなどの使い捨て食器を使うことがあるそうです.クリスマスなどにはちゃんとした食器らしいのですが.ふだんは「家族の団欒(だんらん)の時間を作るため」といっています.

鈴木友子氏

友子 う～ん,それは不思議な感覚ですねぇ…….後はスタートアップで凄いことをしている人達やスーパー大金持ちが身近にいるらしいのですが,外観とか雰囲気とか見たところだけではわからないことですか.一財産築いたからといって着るものを変えたりしないみたいですね.これも日々感じるギャップの一つでしょうか?

トニー 一儲けしたからといって,次の日から紫のスーツでランボルギーニで出社して偉そうにする……というのはシリコンバレーのスタイルではないですよね(笑).

友子 でも,若いエンジニアで早く一儲けしてリタイヤしたい……という人が多いのですが,これはなぜでしょう?

敦 そうそう,株とかに非常に興味をもっているエンジニアが多いですよね.

トニー う～ん,やっぱりシリコンバレーの仕事は,会社の大小に関わらず個人的な犠牲が多いと感じる人が多いようです.競争が激しいですから.だから早く一儲けしてもう少し楽な仕事につきたいという人が多いのだと思います.典型的なパターンはスタートアップの株で一儲けしてリタイヤですね.

敦 役所の仕事にも差があるのにはびっくりしました.ビザを取るためにアメリカ大使館とやり取りしたのですが,非常に厳しいものでした.しかし,こちらに引っ越してから車のナンバープレートをもらうためにDMV(Department of Motor Vehicles)とやり取りしたのですが,これはルーズだったし仕事もアバウトなところがありました.

トニー お役所も連邦政府系とDMVのようなカリフォルニア州の地方系ではまったく管轄が違うし,仕事の進め方も大きな差がありますよね.

☆仕事について

トニー 日本企業にお勤めですが,仕事の進め方とか何か違いはありますか?

友子 電車通勤でないことが大きいですね.だから終電を気にせずついつい夜遅くまで仕事をしてしまうため,運動不足になることがあります.あとは,日本とのやり取りの仕事が多いので,こちらの午後5時がちょうど日本の始業時間だからまた夕方から仕事という感じもあります.

敦 時間的には,シリコンバレーのほうが自己調整的な部分が多く,もう少し自由に調整できるというメリットがあります.また,シリコンバレー現地のエンジニアから,日本側に対して確認することなどが多いのですが,意味やニュアンスが伝わらな

Silicon Valley

H. Tony Chin

対談編

いとか，ちょっとした認識の違いから日米間の間に挟まれることがあります．まあ，仕事柄そういうことを調整するのも役割の一つですが．

友子 アメリカ国内の顧客やパートナー企業とはよくスピーカフォンを用いたテレカンファレンスを行います．出向く手間が省けるので非常に効率的だと思います．必要な人が集まったり，あまり服装などを気にしないでよいので便利です．

トニー そうですね．アメリカ企業だとはっきりとした必要性がないと出向いて打ち合わせがありませんよね．デモなども最近はWebでできますし．でも，テレカンファレンスの相手が見えないからついついほかの仕事をこっそりしたり，バクバクとオヤツを食べたりする人もいますよね(笑)．

さて，毎日の仕事面で苦労したことなどはありますか？

友子 まあ，苦労とまではいえないのですが，やはり言葉の問題でしょうか？　社内で人とすれ違うと挨拶をするのですが，けっこうずっと話し続ける人とかがいますよね．まあ，話の内容も日本と違うから面白いんですが．

敦 日本だと天気の話などをするのですが，シリコンバレーだと，まず何を話してよいのかわからない……．

トニー アメリカ人だとけっこう具体的に「週末に，やれ何をどこで誰とやった」とかこと細かに話しますよね．

友子 そうそう，それもスーパーのレジの人とか，知り合いでもまったくない人にもやるからビックリします．面白い文化だとは思うのですが，はじめは戸惑います．

敦 日本だと他愛もなく当り障りの少ない，天気の話などをするのですが，まず何を話してよいのかわからないので困ります．

友子 私の苦労話は，方向音痴になることです．これは，仕事で打ち合わせを行う場合，相手先のオフィスあるいは，自社オフィスで集合するわけですが，私は高速道路の出口を間違えたりして打ち合わせ時間ぎりぎりまで道に迷ってしまったり，相手先から自社までの道を聞かれたときに，最短の道案内ができなくて少し遠回りをさせてしまったことがあります(苦笑)．

敦 私は苦労話ではないのですが，英語の話があります．私は英会話が不得意なままこちらに来ましたが，仕事柄社外のパートナー企業とコミュニケーションする必要があります．これは，仕事の履歴を残すために，ほぼすべてメールで行っています．英語にはあまり敬語などがないので，意識しないでとりあえず書いたメールでもコミュニケーションができているので助かっています．これが日本語だったりすると気を使ったりするので難しくなるでしょう．

☆アメリカ人はそれほど家電に興味がない?!

敦 イメージと大きくずれていたのは，家電のことでしょうか？　技術の最先端の街というイメージだったのですが，携帯電話，BBなどのインターネットの環境，家電の種類からすると日本のほうがはるかに製品の種類も豊富だし，先行していると感じました．

友子 そうそう，携帯電話とかゴツイしがっかりしました……．

敦 仕事柄コンシューマ向けのプリンタなどの組み込みが多いので，家電にも非常に興味があるのですが，非常に古い機種や日本と比べて機能がはるかに少なくシンプルな製品しか出回っていません．最新の機種を欲しがったり，求めたりしないのでしょうか？

鈴木　敦氏

トニー そうですね，私も長い間アメリカに住んでいるのですが，はっきりとした理由が見えません．たしかに物持ちが良いので日本のような買い換え需要がそれほど多くないのは確かです．つまり壊れるまで使うという意識が強いのです．街の家電修理屋さんがまだありますから．また新聞の個人広告欄で家電を売ったり，ガレージセールで売ったりすることが多いですよね．

友子 紙とかは大量に消費するのに不思議ですね．

トニー テレビとか家電を捨てるのにアメリカでは市によって規制があるので一苦労しますよ．ちゃんとリサイクルの流れができるともう少し買い替えがはやるかもしれませんが……．もう一つ考えられる理由に，アメリカ人の多くがほかの消費にお金を回していることだと思います．つまり，旅行とか趣味にかなりお金をつぎ込んでいるのではないでしょうか．

友子 なるほど，エンジニアでも非常に多趣味で，たとえばヨットをもっているとかいう人がいますよね．給料のすべては趣味につぎ込んでいるとか？

トニー そういう意味では個人差が多いので，なかなか平均的なアメリカ人の懐具合とか消費のパターンとか知るのは難しいのではないかと思います．

トニー・チン　htchin@attglobal.net　WinHawk Consulting

対談を終えて(対談者より)：

不思議な縁でトニーさんと知り合い，二度目にお会いしたのが，この対談の場だった．シリコンバレーの新参者二人にどんな話ができるのか？と思っていたが，実際対談は，こちらの疑問にトニーさんが答えるという，コラムの筆者対読者のような場面が多かったように思う．こちらでの生活が長いトニーさんだが，もともと関西出身ということで，アメリカ文化/習慣などの話題では，われわれの感覚と重なる部分も多いように感じられた．機会があればまたぜひお話をお聞きしたい．(鈴木　敦&鈴木友子)

HARD WARE

●ソフトコアプロセッサ

Nios プロセッサ バージョン 3.0

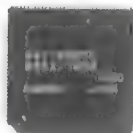
- Cyclone と Stratix デバイスの双方に搭載されている大容量デュアルポートメモリを活用した，ユーザーコンフィグレーション可能な命令用およびデータ用の1次キャッシュを搭載.
- 100MHz 以上の速度で低コスト SDRAM デバイスへのシングルサイクルアクセスを実現することが可能.
- ユーザーコンフィグレーション可能な1次キャッシュと SDRAM コントローラの組み合わせにより，低コストなオフチップメモリを利用し，高性能オンチップ SRAM からの起動とほぼ同等の大容量メモリと性能を得ることが可能.
- パラメータ化されたインターフェースバスの Avalon スイッチファブリックは，データ転送のボトルネックを除去するためのパイプラインデータ転送の処理をサポート.

■日本アルテラ（株）
価格：$495
TEL：03-3340-9480

●32 ビット CPU

H8SX/1650

- 0.18μm の CMOS プロセスを使用し，最大動作周波数は 48MHz 動作で，従来品と比較して 1.5倍の高速化を実現.
- バス幅が2倍の 32 ビットになったことと合わせ，処理能力が約3倍に向上.
- 乗算器の性能向上や除算器内蔵により，48MIPS の高速処理を実現.
- 周辺デバイスとの接続性を容易にするためのリトルエンディアンへの変換機能や，外部メモリのエリアごとに設定可能なアドレス/データのマルチプレクスバス I/O インターフェースを内蔵.
- 新規命令やアドレッシングモードの追加，アドレス空間の拡大により，システムの高機能化にともなうプログラムやデータ容量増大に対応可能.

■（株）日立製作所
サンプル価格：¥900　TEL：03-5201-5276
URL：http://www.hitachisemiconductor.com/jp/

●8 ビット CPU

S1C88649

- 日本レコード協会規格 JIS RIS506-1996（11 × 12 ドットサイズフォント，JIS 漢字第1，第2水準）のアドレスに準拠した漢字フォントを搭載.
- フォントデータは同社のオリジナルフォントを採用.
- 非漢字のフォントおよび Music shift JIS フォントを採用.
- 液晶ドライバ 80 × 16 ドット出力により，6文字の漢字表現が可能.
- 低消費電流は，32kHz Half モードで 2.5μA を実現.
- 1チャネルクロック同期/調歩同期のシリアル通信機能で，本体との接続を実現.

■ セイコーエプソン（株）
サンプル価格：¥800
TEL：042-587-5816
URL：http://www.epsondevice.com/

●USB2.0 トランシーバ

EZ-USB TX2

- USB2.0 トランシーバマクロセルインターフェース仕様に準拠した，スタンドアロン型高速 USB2.0 トランシーバ.
- スキャナ，プリンタ，ディジタルビデオカメラ，ディジタルカメラなどに使用する8ビットまたは16ビットのインターフェースを搭載.
- 複雑な高速アナログ USB コンポーネントをディジタル ASIC に外付けすることが可能となり，開発時間とリスクの減少を実現.
- 56 ピン SSOP または 48 ピン FBGA という2種類のパッケージオプションを用意.

■ 日本サイプレス（株）
価格：¥130（100,000 個単位）
TEL：03-5371-1921　FAX：03-5371-1955

●画像処理プロセッサ

Ri10 シリーズ

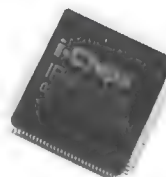
- 独自の並列処理エンジン Ri10 コアを内蔵した，高性能画像処理プロセッサ.
- 224 個のプロセッサエレメントをもつ専用の Ri10 コアを内蔵することで，内部動作周波数 230MHz で 25.7GOPS という性能を達成.
- 高精度・高速画像処理アプリケーションで必要な画像補正，拡大/縮小，階調などの処理をソフトウェアにより実現.
- ミドルウェアの作成は専用のソフトウェア開発支援ツールを使用し，作成した画像処理アルゴリズムをリアルタイムに評価することが可能.

■（株）リコー
サンプル価格：¥15,000 ～ ¥30,000
TEL：045-477-1703
E-mail：lsi-support@ricoh.co.jp
URL：http://www.ricoh.co.jp/LSI/

●画像処理 LSI

IP00C111

- 8 ビット画像処理データに対して，最大 1024 × 1024 画像領域までヒストグラム処理が可能.
- 12 ビット画像データに対して最大 512 × 512 画像領域まで，水平，垂直方向同時プロジェクション処理が可能.
- 水平方向のみ，あるいは垂直方向のみの場合は，それぞれ 1024 画素または 1024 ラインまでプロジェクション処理が可能.
- 8 ビット画像データに対して 3 × 3 局所領域をソーティングし，最大値，中央値，最小値のいずれかを出力可能.
- RGB24 ビットの画像データに対して，3 × 3 マトリクスの係数を任意に設定し，色変換することが可能.

■ アイチップス・テクノロジー（株）
価格：下記へ問い合わせ
TEL：06-6492-7277　FAX：06-6492-7388

HARD WARE

●画像圧縮/伸張 LSI

RB5C634

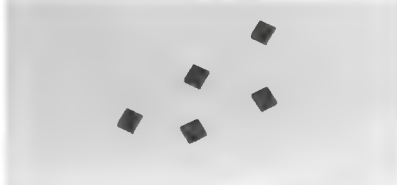
- JPEG2000のエンコード/デコード処理プロセスである画像変換から算術符号化部分，パケット生成，解析機能までを1チップ化．
- JPEG2000のエンコード/デコードが容易に行えるだけでなく，VGAサイズの動画をリアルタイムで毎秒30フレーム処理することが可能．
- 独自のパラメータ制御により，従来困難であった圧縮後のデータ量の制御を画質の劣化を最小限にとどめながら実現．
- 固定長の圧縮データを高画質で出力することが可能であり，秒30枚すべての画像に対して個別にレートコントロールすることが可能．

■（株）リコー
サンプル価格：¥15,000
TEL：045-477-1699
E-mail：lsi-support@ricoh.co.jp
URL：http://www.ricoh.co.jp/LSI/

●昇圧型スイッチングレギュレータコントローラ

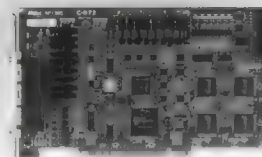
S-8355/57シリーズ S-8356/58シリーズ

- 小型SNBパッケージを採用し，スイッチング周波数600kHzの高周波版昇圧型スイッチングレギュレータコントローラ．
- コイルに保持するエネルギを小さくすることで，従来の300kHz品と比較してコイルの小型化が可能．
- 小型6ピンSNBパッケージ（1.8×2.0×0.8mm）の採用により，実装面積が半分以下になり，携帯機器の小型化に寄与．
- V_{DD}/V_{OUT}分離型はIC内の回路電源と出力電圧値を決めるV_{OUT}端子が分離されており，出力電圧値を外付け抵抗により可変したい，出力電圧値を高くしたいといった用途に適する．

■ セイコーインスツルメンツ（株）
サンプル価格：¥150
TEL：043-211-1193

●ステッピング/サーボモータ用8軸コントローラー

C-872

- 8軸/1スロットの実現により，多軸制御でのスロット不足を解消し，軸あたりのコスト削減が可能．
- 同社製のチップコントローラを搭載し，インテリジェント機能により，各ドライブは簡単なコマンドで制御が可能．
- ソフトウェアの互換性，オプションケーブルにより，ハードウェアの互換性を維持．
- デバイスドライバソフトおよびケーブル，分配ボードなど，製品の立ち上げを容易にするオプション製品を用意．
- コンペアレジスタ付きカウンタにより，エンコーダなど外部パルスを入力し，閉ループ制御，指定位置でのトリガ信号などとして利用可能．

■（株）メレック
価格：下記に問い合わせ
TEL：0426-64-5383

●乾電池漏液対策用ディテクタIC

S1F77600

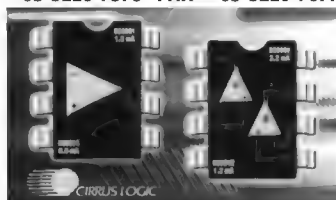
- 乾電池2セル直列電源の電圧降下の検出．
- 低電圧動作，電源電圧0Vまで出力を保証．
- 高精度で低消費電流を実現．
- 検出電圧は，高精度のため調整不要．
- 基準電圧回路，コンパレータ，電圧検出用抵抗，ヒステリシス回路などを内蔵．

■ セイコーエプソン（株）
サンプル価格：¥100
TEL：042-587-5816
URL：http://www.epsondevice.com/

●OPアンプ

CS3001/02/11/12

- 低周波のシグナル設計用途に適しており，設計のアップグレードが簡単にできる業界標準のパッケージピン配列で，高い性能を実現するよう設計されている．
- 産業計測用A-Dコンバータ製品シリーズで開発，実用化した，独自のMultiPath技術を採用．
- 低雑音（6nV/$\sqrt{Hz}$ @0.5Hz，1/*f*ノイズコーナー@0.08Hz），低ドリフト（最大で0.05μV/℃）および高いオープンループゲインを提供．
- オープンループゲインは，1000兆（300dB）のオープンループゲインのOPアンプに近いものとなっている．

■ シーラス・ロジック（株）
サンプル価格：$1.94（CS3001/11）
$3.02（CS3002/12）
TEL：03-5226-7378 FAX：03-5226-7677

● Ethernetスイッチングチップ

ZL50418ファミリ

- Ethernet集束装置の幅広いニーズに対応しており，16個の高速Ethernetポートを2個のギガビットEthernetポートと統合し，最大7.2Gbpsのノンブロッキングスループットを可能にする．
- スイッチは，ワイヤスピードで最大10.712Mppsの転送が可能．
- ギガビットポートは，レイヤ2転送とレイヤ3/4クラシフィケーションを提供し，高速Ethernetポートとの連携に対応するようにプログラム可能．
- 最大八つの出力キューがあり，サービスプロバイダはその出力キューを，優先レベルに基づいてトラフィック識別するようにプログラムできる．
- 八つの出力キューは，IETFのDiffscrvセットで定められている八つのトラフィッククラスに分けられると同時に，これをフルサポートしている．八つのクラスには，ネットワーク管理，優先転送，確定転送，ベストエフォートクラスなどがある．

■ ザーリンク・セミコンダクター・ジャパン（株）
価格：$79.73
TEL：03-5733-8181 FAX：03-5401-2441

HARD WARE

●DC-DCコントローラ ドライブチップセット

LM5041-LM5102チップセット

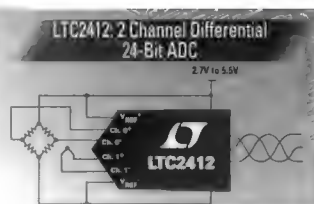
- コンピュータや通信システムの電圧変換に最適な集積型高電圧カスケードDC-DCコントローラ.
- デッドタイムがプログラム可能で，同期降圧制御とプッシュプルドライバを集積.
- ワイドレンジの15～100Vスタートアップバイアスレギュレータ搭載.
- ピーク電流2Aの高速プッシュプルMOSFETドライバ.
- ユーザープログラマブルな，ソフトスタート回路.
- 小型のサーマルエンハンスト16ピンリードレス/リードフレームパッケージを採用.
- 100V同期降圧ドライバに対応したカスケードコントローラ.
- フォワードインタリーブト，カスケードプッシュプル，ハーフブリッジまたはフルブリッジなど機能の柔軟な構成が可能.

■ ナショナル セミコンダクター ジャパン(株)
価格：下記へ問い合わせ
TEL：03-5639-7488

●24ビットA-Dコンバータ

LTC2412

- 自動チャネル選択機能付き2チャネル差動入力.
- SPIプロトコルと互換性のある3線インターフェースをもつ.
- INLは2ppmで，ミッシングコードはなし.
- ノイズは800nV_{RMS}，フルスケール誤差は2.5ppmでオフセット誤差は0.1ppmを実現.
- 発振器を内蔵しているため，外付け部品が不要.
- 200μAの低消費電流を実現．オートスリープ時は4μA.
- 最小110dB，50Hz/60Hzのノッチフィルタをサポート.
- 2.7V～5.5Vの単一電源動作.

■ リニアテクノロジー(株)
サンプル価格：¥570
TEL：03-5226-7291　FAX：03-5226-0268

●リアルタイムOS付きCPUボード

OAKS32シリーズ

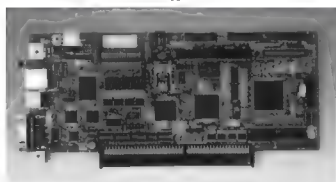
- 三菱電機製16/32ビットマイコンM32C M30833FJ/M30835FJを搭載.
- 24MIPSで，動作周波数は32MHz.
- 内蔵フラッシュメモリを512Kバイト，RAMを31Kバイト搭載.
- CコンパイラNC308，デバッガKPD308およびフラッシュメモリライタ付属.
- 拡張ボード，回路部品，電源アダプタ付きのフルキットを用意.
- オプションとして10Base-T LANボード，LCD表示ボードを用意.
- 各種工業製品への組み込み，FA，検査装置などへの応用，マイコン評価，学習などの用途に適する.

■ アンカーシステムズ(株)
価格：¥9,800～
E-mail：anchor@noritz.co.jp
URL：http://www2.noritz.co.jp/anchor/

●マイコン用ソフト開発プラットホーム

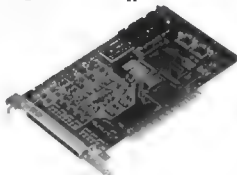
SolutionGear-MINI

- ソフトウェアツールとして添付の同社製のコンパイラおよびデバッガの利用が可能.
- 評価版のRTOS(μITRON準拠)，ミドルウェア(TCP/IP，Webサーバ，PPP，DNS，FTPサーバなど)，デバイスドライバなどをすべて標準添付.
- Ethernet，USB，フラッシュカードI/Fをそれぞれ搭載しており，単体でインターネット接続アプリケーションの先行開発が可能.
- 搭載デバイスのサンプルドライバが添付されているため，すぐに評価を開始できる.
- CPUに搭載されたオンチップデバッグ機能により，N-Wire接続のインサーキットエミュレータと接続して高度なデバッグが可能.

■ NECエレクトロニクス(株)
サンプル価格：¥250,000
TEL：044-435-9494
E-mail：info@lsi.nec.co.jp

●パソコン計測制御コンポーネント

ADA・Fシリーズ

- アナログ入力(分解能16ビット・32チャネル)，アナログ出力(分解能16ビット・2チャネル)，ディジタル入出力，カウンタ機能を搭載.
- アナログ入出力の動作の開始/停止/クロック制御は，各機能のイベントや外部制御信号入力を自由に組み合わせられるイベントコントローラによってハードウェアで統合管理されているため，ソフトウェアに依存しない各機能間の同期制御を行うことが可能.
- アナログ入力と出力は，個々または同時にバスマスタ転送を行うことができ，CPUに負荷をかけることなくパソコンに大容量のデータを転送することが可能.

■ (株)コンテック
価格：¥108,000(PCIバス対応)
¥88,000(CardBus対応)
TEL：03-5628-9286　FAX：03-5628-9344
E-mail：tsc@contec.co.jp

●デザインキット

ARM9デザインキット

- クロスツール，RTOS，ミドルウェア，評価ボードをセットにしたパッケージ製品.
- 評価ボードは，デスクトップPCの空きスロットに挿入して評価，試験を行う.
- 試作評価に必要な機能として「MPUまわりの消費電力測定，表示」，「実行時間の測定，表示」を標準装備.
- 量産時の目標性能測定，ソフトのベンチマーク試験などに有効.
- クロスツールは，使用制限なしのフルバージョンで提供.
- フレームワーク，C/C++コンパイラ，アセンブラ，シミュレータに加え，新開発の評価ボードのPCIバスを直接制御して，ROM上のデバッガI/Fを利用するリモートデバッガを装備.
- リアルタイムOSとして，MISPO社製のNORTi 4.0のカーネルを標準装備.
- μITRONのタスク遷移をモニタする「タスクステートウォッチャ」機能を装備.

■ ガイオテクノロジー(株)
価格：¥298,000
¥398,000(TCP/IP付き)
TEL：03-3662-3041　FAX：03-3662-3043
E-mail：sales@gaio.co.jp

HARD WARE

●フラッシュカード/ディスクモジュール

ATA フラッシュカード/コンパクトフラッシュカード/IDE フラッシュドライブ/フラッシュディスクモジュール

- 米国シンプル・テック社が開発した，フラッシュカードおよびディスクモジュール．
- ATA フラッシュカードは，32M バイトから 4G バイトまで 16 種類に対応．対応電圧は 3.3V，5V で，PCMCIA Release2.1 に対応．書き込み速度は最大 3M バイト/s，読み込み速度は最大 1.5M バイト/s を達成．対応温度は，0～70 ℃，－25～85 ℃，－40～85 ℃の 3 種類に対応．
- コンパクトフラッシュカードは，32M バイトから 1G バイトまで 10 種類に対応．対応電圧は 3.3V，5V で，PCMCIA Release2.1 に対応．書き込み速度は最大 3M バイト/s，読み込み速度は最大 1.5M バイト/s を達成．対応温度は，0～70 ℃，－25～85 ℃，－40～85 ℃の 3 種類に対応．

■（株）アスク
価格： ATA フラッシュカード ¥738,100（4G バイト）
¥165,500（1G バイト）
コンパクトフラッシュカード ¥206,000（1G バイト）
¥44,400（256M バイト）
IDE フラッシュドライブ ¥738,700（4G バイト）
¥187,200（1G バイト）
フラッシュディスクモジュール ¥101,100（512M バイト）
¥48,300（256M バイト）
TEL ： 03-5215-5650　FAX ： 03-5215-5651
URL ： http://www.ask-corp.co.jp/

● PXI 用高周波数発生器

PXI-5404 100MHz 高周波数発生器

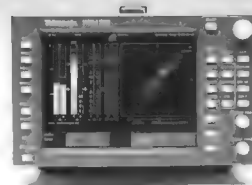
- 正弦波とクロックを同時に出力，± 0.2dB という正弦波レベル精度により，自動テスト装置（ATE）のコヒーレントサンプリング，機能テストや設計評価でのアプリケーションに適する．
- 周波数ソースでは，$2V_{pp}$ から $1V_{pp}$ まで調整可能な正弦振幅，12 ビットの垂直分解能，± 0.2dB というフラットな正弦波通過帯域を持つ 9kHz～100MHz までの正弦波と，DC から 100MHz までのクロックを発生可能．
- PLL により，PXI バックプレーンや高速ディジタイザのような他のデバイスと同期を取ることが可能．
- IVI に準拠する計測器ドライバ NI-FGEN が付属しており，LabVIEW，LabWindows/CVI，Visual Basic および Visual C++ のフルプログラミングサポートを提供．
- NI-FGEN ソフトウェアで利用可能なシミュレーションモードにより，システム内にハードウェアがない場合でもアプリケーション開発とプロトタイプの作成を行うことが可能．

■ 日本ナショナルインスツルメンツ（株）
価格： ¥210,000
TEL ： 03-5472-2970　FAX ： 03-5472-2977
E-mail ： prjapan@ni.com

●波形モニタ用オプション

WFM700 シリーズ

- WFM700 型 SD/HD マルチ波形モニタシリーズ用のオーディオオプション．
- オーディオ信号とビデオ信号の品質，内容，適合性の監視，検証を 1 台で実現．
- ステレオ，5.1 チャネル，7.1 チャネルのマルチチャネルオーディオに対応．
- AES/EBU 入力またはディジタルビデオ信号内のエンベデッドオーディオ入力のレベルバー，リサージュ表示，チャネルステータス表示や CRC エラーチェックなどの機能をサポート．
- ファームウェアアップグレードにより，これまでのマルチフォーマットビデオの機能が拡張され，ステータスレポートおよびエラー検出の機能を追加．

■ 日本テクトロニクス（株）
価格：下記へ問い合わせ
TEL ： 03-3448-3010　FAX ： 0120-046-011
URL ： http://www.tektronix.co.jp/

●ファンクション/任意波形発生器

33220A ファンクション/任意波形ジェネレータ

- 最高周波数 20MHz を実現し，64 キロポイントによる任意波形生成，パルスを含む波形モード，ディジタル試験を可能にする可変エッジ時間パルスとパルス幅変調などの性能を実現．
- パソコンと LAN 経由で接続した場合，遠隔地から Web ブラウザでフロントパネルを疑似的に操作することが可能．
- GPIB に加え，USB と Ethernet のインターフェースも標準装備．
- ダイレクトディジタルシンセシス技術を採用することにより，高精度で安定した出力信号を低価格で提供．

■ アジレント・テクノロジー（株）
価格： ¥224,000
TEL ： 0120-421-345
URL ： http://www.agilent.co.jp/

● Bluetooth 認証テスタ

BITE RF テスター BITE プロトコル/プロファイルテスター

- スペインの CETECOM 社が開発した，Bluetooth 認証テスタ．
- BITE RF テスターは，Bluetooth の RF 試験のテストケースに適合した，多機能な自動統合検査システム．
- BITE プロトコル/プロファイルテスターは，認証テストケース参照リストに準拠した，Bluetooth のプロトコルとプロファイルの適合性，相互接続性を自動で検査する総合システム．
- コンテンツ目録，日付，テストケースのカテゴリ分け，地図化などの情報処理といった認証プロセスを効率的かつスピーディに行うことができる．すべての操作を Windows2000 環境で行うことができ，パソコン上の Word や Excel にデータを出力し，レポートを自動作成するなどの機能を搭載．

■ コーンズ・アンド・カンパニー・リミテッド
価格： ¥75,000,000
（BITE RF テスター）
¥25,000,000
（BITE プロトコル/プロファイルテスター）
TEL ： 03-5730-1634
FAX ： 03-5730-1622
E-mail ：
system@tky.cornes.co.jp

●青色光対応光パワーメータ

TB100

- 波長 405nm の青色レーザの光パワー測定に適する．
- 400nm～420nm の範囲で波長感度差が約 2%であるため，波長に依存しない正確な測定が可能．
- センサは，フィルタ表面に反射防止処理を施しているため，外部の反射が多い場合でも安定した測定が可能．
- センサ受光径は Φ 18mm と大きく，厳密に距離を調整しなくても，安定した測定が可能．

■ 横河電機（株）
価格： ¥198,000
TEL ： 0120-137046　FAX ： 0422-52-6624

SOFT WARE

●ソフト開発ツール

DSP56xxx用 ソフト開発ツール

- モトローラのDSP56xxxアーキテクチャをサポートするソフトウェア開発ツール.
- DSP56366, DSP56367, DSP5637xやDSP5637xEVMなどの最新のDSP56xxxシリーズの派生品と評価ボードをサポート.
- CrossView Proデバッガのダイアログは, 柔軟性の高いブレークポイントの設定やプローブポイントの設定を可能にし, I/Oシミュレーションをより使いやすいものにしている.
- ソフトウェア開発に必要なすべての機能を統合開発環境EDEのもとで動作させることができ, C/C++, EC++コンパイラ, アセンブラ, リンカ, ロケータ, CrossView Proシミュレータデバッガで構成される.
- DSP56xxx評価ボードを用いるためのCrossView Pro ADS/EVMデバッガの供給が可能.
- Windows, LinuxおよびSolarisプラットホーム用を提供.

■ アルティウム ジャパン(株)
価格: 下記へ問い合わせ
TEL : 03-5436-2501 FAX : 03-5436-2505
E-mail : info@altium.co.jp

●DOS互換ファイルシステム

PrFILE for Memory Stick

- メモリースティックPROに対応した, DOS互換FATファイルシステム.
- メモリースティックPRO対応ホストコントローラIP用のサンプルドライバおよびアクセスライブラリを同梱.
- 動画などの大容量データの高速書き込みを実現.
- CPU/OSに非依存で, さまざまなOS上, あるいはOSのない環境で動作が可能.
- FAT12/16/32およびVFATに対応したDOS互換のファイルシステム.
- 同社製RTOSベースシステム向けの開発スイート「eBinder」を利用すれば, アプリケーションの短期間, ローコストでの開発を実現. 内部構造を詳細に知らなくても, 内部構造をわかりやすくホストGUI画面で表示, 操作可能なコンフィグレーションツール, システム分析ツールを同梱.

■ イーソル(株)
価格: 下記へ問い合わせ
TEL : 03-5301-5325 FAX : 03-5376-2538
E-mail : ep-inq@esol.co.jp
URL : http://www.esol.co.jp/embedded/

●通信関連製品設計検証用EDAソフトウェアー

アドバンスド・デザイン・システム 2003A

- 電磁界モデリングを使用した任意形状のパラメタライズド部品の作成, 登録ができる「Advanced Model Composer」機能を搭載.
- 一度の電磁界シミュレーションで任意に形状変化が可能な電磁界モデルの作成, ライブラリ登録が可能となり, レイアウトの検証時間を大幅に短縮することが可能.
- 回路のビヘイビアモデルを自動的に作成し, システムレベルで高速検証を行うことができる「Automatic Behavior Modeling」を搭載.
- W-CDMA ACLR測定のシミュレーションの場合, シミュレーション時間を最大で1/170にまで短縮可能.
- ケイデンス・デザイン・システムズのEDAツールのフレームワークとの統合が可能となっている.
- 多数のファウンドリ対応のデザインキットを提供.
- 5GHzワイヤレスLANや, 次世代通信規格対応などの豊富な通信システムライブラリを用意.

■ アジレント・テクノロジー(株)
価格: ¥1,000,000～
TEL : 0120-421-345

●リモートアクセスアプライアンスサーバ

Virtual Office Server

- シトリックス社のCitrix Secure Gateway技術と, VMware社のVMware GSX Server技術を融合した, インターネットリモートアクセスアプライアンスサーバ.
- 従来は3台の物理サーバを必要としていたが, VMware GSX Serverとの連携により, 1物理サーバにパッケージすることを可能にしている.
- サーバベースコンピューティングのための高度なセキュリティ製品で, MetaFrameサーバの認証機能とSSL暗号機能を備えたシトリックス社のCSGと, 複数のサーバOSをソフトウェア的にパーティショニング化.
- 現状のインターネットリモートアクセスの問題点であるVPNクライアントインストールのわずらわしさやVPNコンフィグレーションの複雑さなどを解消.
- アプリケーションコントロールや, ワンタイムパスワード個人認証機能を搭載.

■ (株)ネットワールド
価格: ¥2,100,000 (TypeA, MetaFrameなし)
¥2,800,000
(TypeB, MetaFrame5ユーザーバンドル)
TEL : 03-5210-5081
E-mail : vos-info@networld.co.jp

●総合画像圧縮, 解凍, 編集ライブラリ

ImagXpress Professional

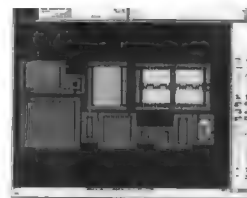
- 米国ペガサスイメージング社が開発した, 静止画, 動画の取り込み, 編集, 圧縮, 解凍, 書き出しを可能とするデスクトップアプリケーションおよびカスタム製品の開発を可能にするライブラリ製品.
- プログラム開発環境は, Win32ビジュアル開発環境を採用.
- .Net, Visual Basic, Delphi, VC++, C++ Builder, HTML, Accessのサンプルコードを提供.
- ATLを使用することで, 最小限のフットプリントを実現し, MFCを必要としない.
- ActiveX/COMコントロールをホストとするあらゆる開発環境での使用が可能.
- スキャン, エディット, 変換, 表示, 印刷などの各種機能をサポート.
- クライアント/サーバWeb開発機能を搭載.
- ファイル, メモリバッファ, インターネットアドレスからの30を超える画像フォーマットをサポート.

■ エクセルソフト(株)
価格: ¥139,000 (1開発者ライセンス)
TEL : 03-5440-7875 FAX : 03-5440-7876
E-mail : xlsoftkk@xlsoft.com
URL : http://www.xlsoft.com/

●アナログレイアウト設計支援ツール

AMPER

- 回路図の配置情報やネットリスト, あるいは既存のレイアウトより配置制約を抽出し, 新規レイアウトの配置に適応させることが可能.
- 一括実行だけでなく, New-SXのエディタ内においてインタラクティブに使用することも想定しており, レイアウト設計のさまざまな場面で設計者を支援.
- 回路図の配置を活かした初期配置.
- 既存レイアウトの配置を使った同形配置.
- Windows準拠のGUIにより, 制約設定を容易化.
- 部分コンパクション機能を搭載.

■ セイコーインスツルメンツ(株)
価格: ¥9,000,000
TEL : 03-5645-1741

SOFT WARE

●3次元CADソフトウェア

SolidMX V1.0

- 線画を直接3次元空間上に書き込めるために，試行錯誤が必要な構想検討段階のあいまいさを含んだ形状でも柔軟な設計が可能な3次元CAD.
- 構想検討段階で作成した線画をもとに，サーフェスモデルやソリッドモデルの作成が可能.
- 従来は2次元CAD上で行っていた，設計者の意図や製造部門への加工指示などの情報の記入を，3次元CAD上でも行える3Dドラフティング機能を装備.
- 面や線をマウスで自由に形状を編集できるダイレクトモデリング手法を採用.
- 複数のモデルをパラメータの設定で一括して編集したい場合は，フィーチャーベースパラメトリック手法の利用が可能.

■ 富士通(株)
価格：¥1,190,000
TEL：043-299-3233
E-mail：mx110@fes.fujitsu.com

●設計業務向けコラボレーションツール

DENZU

- 米国netGuru社が開発した，Webベースのリアルタイムコラボレーションツール.
- 作成したアプリケーションがなくても，2D/3DドキュメントをWebブラウザ上でビューおよびマークアップが可能.
- ドキュメントのズームイン/アウト，指定範囲の拡大，図面の回転，パニング，メジャメントなどのビュー機能を搭載.
- テキスト入力，オブジェクト(線，円，矩形，雲型)の入力，イメージの貼り付け，塗りつぶし，チャットオブジェクトの追加などのマークアップ機能を搭載.
- 会議の議事録として利用可能な，保存/再生機能.
- 2Dや3DなどのCAD図面から，MS-Officeドキュメントやイメージファイルなど，150種類以上の多様なファイルフォーマットに対応.
- 投票用紙の作成，配布，結果集計，結果公表などの投票機能をサポート.

■ (株)マクニカ
予定価格：¥150,000
TEL：045-476-1960　FAX：045-476-1970
E-mail：press@support.macnica.co.jp
URL：http://www.networks.macnica.co.jp/

●電子文書管理ツール

BCL Magellan 6.0

- 米国BCLテクノロジー社が開発した，電子文書管理ツール.
- DOC，XLS，PPT，TXT，BMP，JPEG，TIFF，CHM，PDFなど印刷可能なWindows上の文書をすべてHTMLに変換.
- Adobe Acrobatがインストールされていないシステム上で，PDFを含む電子文書をHTMLに変換できる.
- 標準のデスクトップ画面用に，スタイルシートとNetscapeレイヤを使用してHTMLを表示するため，オリジナル文書と同じレイアウトの高精度な出力を実現.
- ラスタ画像とベクタ画像を正確に変換し，JPEGファイルに保存してオリジナルのサイズを保持.
- Magellan COMオブジェクトを使用して，カスタムアプリケーションの開発，ソリューション提供が可能.
- Magellan Batchプロセスを使用して，複数文書の一括変換が可能.

■ エクセルソフト(株)
価格：¥26,000
TEL：03-5440-7875　FAX：03-5440-7876
E-mail：xlsoftkk@xlsoft.com
URL：http://www.xlsoft.com/

●統合パッケージ

ThreadX/USB統合キット

- Visual Basic，C/C++での開発が可能．米国エクスプレスロジック社が開発したリアルタイムOS「ThreadX」と自社開発のUSBプロトコルスタック「GR-USB/HOST」や「GR-USB/OTG」を統合した製品.
- ロイヤリティフリーの契約.
- ANSI Cでのソースコード提供.
- 15か月間までの技術サポート付き.
- 日本語での技術サポート.
- 各種ポーティング，受託開発も可能.

■ (株)グレープシステム
価格：下記へ問い合わせ
TEL：045-222-3761　FAX：045-222-3759
E-mail：Info@threadx.grape.co.jp
URL：
http://www.grape.co.jp/el/thread_usb.html

●Webアプリケーションフレームワーク

Sledge

- フリーソフトウェアとして提供されるWebアプリケーションフレームワーク.
- HTTPリクエストパラメータ処理.
- CookieやURL埋め込みによるセッション維持機能を搭載.
- MySQLやPostgreSQLなどを利用したセッションの永続化.
- ページコンポーネントやアクション単位でのユーザー認証が可能.
- テンプレートエンジンの自動呼び出し機能をサポート.
- ステージング環境とプロダクション環境での設定ファイルの読み分けをサポート.
- ユーザー入力のバリデーション処理，アプリケーションの動作ログ出力，不要となったセッションの自動消去，トークン機能，Cookieを利用したオートログイン機能などの拡張ライブラリを提供.

■ (株)オン・ザ・エッヂ
価格：フリーソフトウェア
TEL：03-5766-7211
E-mail：press@edge.jp

●データベースエンジン

InterBase 7

- SMP対応により，マルチプロセッサシステムにおける性能を大幅に向上.
- クライアントの安定性の強化および性能の改善を実現するスレッドセーフクライアントライブラリを搭載.
- JavaによるアクセスをJDBC Type4ドライバを搭載.
- アプリケーション開発の配布段階において，一時テーブル経由の接続監視をサポート.
- MGAを装備しているため，独自のバージョニング機能により，短期的なOLTPを効率的に処理するとともに，意思決定支援アプリケーションなどの長期的な並列処理においても，ほかのデータベースをしのぐ性能を発揮.
- アクティブデータベース機能には，特許技術であるインベントアラータ，ストアドプロシージャ，トリガ，UDFおよびBLOBフィルタなどを搭載.

■ ボーランド(株)
価格：¥9,000～¥550,000
TEL：03-5350-9380　FAX：03-5350-9369

海外・国内イベント/セミナー情報

INFORMATION

海外イベント

3/30-4/3 **The 2003 IEEE International Magnetics Conference**
Boston Marriot Copley Place, Boston, MA, USA
IEEE
http://www.intermagconference.com/

4/1-4 **COMDEX CHINA 2003**
China International Exhibition Center, Beijing, China
Key3Media
http://www.key3media.com/international/events/index.php?d=china&s=welcome

4/22-26 **Embedded Systems Conference**
Moscone Convention Center, San Francisco, CA, USA
CMP Media Inc.
http://cmp.iconvention.com/sf/v33/index.cvn?id=10008&stab=2

4/27-5/2 **NETWORLD+INTEROP Las Vegas 2003**
Las Vegas Convention Center, Las Vegas, NV, USA
Key3Media
http://www.interop.com/lasvegas2003/

5/5-8 **SEMICON Singapore 2003**
Suntec Singapore International Convention & Exhibition Centre, Singapore
SEMI
http://events.semi.org/semiconsingapore/V33/index.cvn

5/11-15 **International Conference for Communications**
Egan Convention Center, Anchorage, AL, USA
IEEE
http://www.icc2003.com/

5/13-16 **Electronic Entertainment Exposition**
Los Angels Convention Center, Los Angels, CA, USA
Electronic Entertainment Expo
http://www.e3expo.com/

国内イベント

4/3-6 **ROBODEX2003**
パシフィコ横浜(神奈川県横浜市)
ROBODEX実行委員会
http://www.robodex.org/

4/9-11 **SEMI FPD Expo 2003**
東京国際展示場(東京ビッグサイト,東京都江東区)
SEMI
http://events.semi.org/semifpdexpo/V33/

4/16-18 **TECHNO-FRONTIER 2003**
日本コンベンションセンター(幕張メッセ,千葉県千葉市)
JAPAN MANAGEMENT ASSOCIATION
http://www.jma.or.jp/tf/

4/16-18 **MICROELECTRONICS SHOW**
東京流通センター(東京都大田区)
(社)エレクトロニクス実装学会
http://www.jiep.or.jp/jiepweb/index.html

4/17-18 **Photomask Japan 2003**
パシフィコ横浜(神奈川県横浜市)
(財)日本学会事務センター内ホトマスクジャパン2003事務局
http://edpex104.bcasj.or.jp/pmj/exhibi_j_fr.html

5/14-16 **2003実装プロセステクノロジー展**
日本コンベンションセンター(幕張メッセ,千葉県千葉市)
(社)日本ロボット工業会
http://protec.jesa.or.jp/jp/frontpage/index.html

5/21-23 **LinuxWorld Expo/Tokyo 2003**
東京国際展示場(東京ビッグサイト,東京都江東区)
IDGジャパン
http://www.idg.co.jp/expo/lw/

開催日,イベント名,開催地,問い合わせ先の順

セミナー情報

C言語によるはじめてのLinuxプログラミング
開催日時　:4月3日(木)~4月4日(金)
開催場所　:DISパソコンスクール(東京都文京区)
受講料　:92,000円
問い合わせ先:(株)エイチアイICP事業部,☎(03)3719-8155,FAX(03)3793-5109 http://icp.hicorp.co.jp/seminar/linux/clinux.asp

高速時代のノイズ対策とプリント基板設計技術
開催日時　:4月4日(金)
開催場所　:CQ出版セミナールーム
受講料　:13,000円
問い合わせ先:エレクトロニクス・セミナー事務局,☎(03)5395-2125

Linux開発環境構築
開催日時　:4月7日(月)
開催場所　:DISパソコンスクール(東京都文京区)
受講料　:46,000円
問い合わせ先:(株)エイチアイICP事業部,☎(03)3719-8155,FAX(03)3793-5109 http://icp.hicorp.co.jp/seminar/linux/linuxtool.asp

JTAGを用いた電子回路基板テスト法
開催日時　:4月8日(火)~4月9日(水)
開催場所　:高度ポリテクセンター(千葉県千葉市)
受講料　:30,000円
問い合わせ先:雇用・能力開発機構 高度ポリテクセンター事業課,(043)296-2582 http://www.apc.ehdo.go.jp/seminar/jyohousub_out/syousai/02IIW22065.html

VoIPとネットワーク構築入門
開催日時　:4月11日(金)
開催場所　:CQ出版セミナールーム
受講料　:13,000円
問い合わせ先:エレクトロニクス・セミナー事務局,☎(03)5395-2125

無線LAN/Bluetoothの規格解説とデモンストレーション
開催日時　:4月16日(水)~4月17日(木)
開催場所　:オームビル(東京都千代田区)
受講料　:68,500円(1口で1社3名まで受講可)
問い合わせ先:(株)トリケップス,☎(03)3294-2547,FAX(03)3293-5831 http://www.catnet.ne.jp/triceps/sem/c030416a.htm

Linuxによるハードウェア制御【入門編】
開催日時　:4月17日(木)
開催場所　:CQ出版セミナールーム
受講料　:13,000円
問い合わせ先:エレクトロニクス・セミナー事務局,☎(03)5395-2125

PC実習!! Windows2000 TCP/IPプロトコル(IPv4,IPv6)&サービス
開催日時　:4月17日(木)~4月18日(金)
開催場所　:SRCセミナールーム(東京都高田馬場)
受講料　:95,000円
問い合わせ先:(株)ソフト・リサーチ・センター,☎(03)5272-6071 http://www.src-j.com/training_no/windows2000_net2.htm

ITRON入門と通信プロトコルスタック
開催日時　:4月18日(金)
開催場所　:みなとみらい2-3-3クイーンズタワーB 7Fクイーンズフォーラム会議室(横浜市西区)
受講料　:無料
問い合わせ先:(株)グレープシステム基本ソフトウェア事業部セミナー係,☎(045)222-3761,FAX(045)222-3759 http://www.grape.co.jp/seminar.html

Delphiビジュアルプログラミング
開催日時　:4月22日(火)~4月24日(木)
開催場所　:高度ポリテクセンター(千葉県千葉市)
受講料　:40,000円
問い合わせ先:雇用・能力開発機構 高度ポリテクセンター事業課,☎(043)296-2582 http://www.apc.ehdo.go.jp/seminar/jyohousub_out/syousai/03IIW20007.html

~大規模FPGA搭載KIT1050 PCIボードを使った~
PCIボード開発者(ハード/ソフト)養成講座
開催日時　:4月23日(水)~4月24日(木)
開催場所　:ちよだ中小企業センター(東京都千代田区)
受講料　:PLXボード付き198,000円(1~3名),セミナー+技術資料148,800円(1名),セミナーのみ68,800円(1名)
問い合わせ先:(株)トリケップス,☎(03)3294-2547,FAX(03)3293-5831 http://www.catnet.ne.jp/triceps/sem/030423a.htm

最新CCDイメージセンサの特性と技術~CCDの基礎と応用技術~
開催日時　:4月24日(木)~4月25日(金)
開催場所　:CQ出版セミナールーム
受講料　:25,000円
問い合わせ先:エレクトロニクス・セミナー事務局,☎(03)5395-2125

バグを出さないための!! ソフトウェア・デバッグ工学とテスト技法
開催日時　:4月24日(木)~4月25日(金)
開催場所　:SRCセミナールーム(東京都高田馬場)
受講料　:76,000円
問い合わせ先:(株)ソフト・リサーチ・センター,☎(03)5272-6071 http://www.src-j.com/seminar_no/23/23_093.htm

日程はすべて予定です.問い合わせ先にご確認のうえ,お出かけください.

IPパケットの隙間から 55

呪いとインフルエンザとLinux初心者

祐安重夫

ちょうどその時，熱が38.5度を超えて，ほとんど起き上がれない状態だった．先月号の原稿のせいで，ヨグ＝ソトース（Yog-Sothoth）の祟りか，ナイアルラトホテップ（Nyarlathotep）の呪いにでもかかったかのような気分である．そんな最中の午後3時過ぎに，その電話はかかってきた．友人のMさんからである．やっぱり呪われている．

こちらはやっと電話のところまで這って行ったところだし，Mさんの話は例によって要領を得ない．どうやらLinuxをインストールして家庭内LANのファイルサーバにしたいらしいのだが，こちらの頭がついていかない．そのうちに気分が悪くなってきたので，後日あらためて電話をもらうことにした．Mさんは「それじゃ今日はもう寝る」といって電話を切った．いい忘れたが，Mさんはアメリカ西海岸に住んでいるので，現地はそろそろ午後11時である．

少し横になった後，意を決して近所の医院までやっとたどりつき，インフルエンザと診断されて薬を出してもらい，昨夜から何も食べていないので点滴を打ってもらった．どうやら，古（いにしえ）えの邪神（Great Old Ones）の呪いではなかったようだ．それから3日ほどは，スポーツドリンクで命をつないでいた．

インフルエンザが治った頃を見計らって，やはり午後3時過ぎにMさんから電話があった．デスクトップにLinuxをインストールし，SambaでWindows用のファイルサーバを作ったらしい．それ自身はまったく問題なく，Windowsから日本語名のファイルを作っても，Linux側でも日本語ファイル名として認識されている．もちろんそれはSambaがWindows側はシフトJIS，Linux側は日本語EUCで見えるように，ファイル名の自動変換をしているからだ．

そこでMさんは，自分のふだん使用しているノートパソコンにもLinuxをインストールしてしまった．そうしてノート側からサーバ側のファイルシステムを，nautilusからSMBプロトコルで見たところ，日本語ファイル名が化けているというのである．もうおわかりかと思うが，Sambaで公開されたファイルシステムをLinuxからSMBプロトコルで見ようとしても，ファイル名はシフトJISに変換されてしまっているから，化けるのは当然である．

UNIX系のOS間でファイルシステムの共有をする場合，NFSと相場が決まっている．そこで，両方のLinuxマシンにNFSをインストールして動かすように説明した．じつはこの時点で，本当にMさんにインストール可能かどうか心配していたのだが，最近のディストリビューション（RedHat8.0だった）はよくできているもので，設定完了まではまったく電話がなかった．

しかし，落とし穴というのはどこにでもあるもので，じつは家庭内LANにはDNSが存在しなかったのだ．もちろん，DHCPでIPアドレスや外部のDNSは指定されるが，これだけではルータの内側についてはDNS検索できるはずがない．つまり，NFSのクライアント側から，サーバの名前を検索する方法がないのである．仕方がないので，クライアント側の`/etc/hosts`に現在のサーバのIPアドレスと名前を記述してもらって，なんとかしのいだ．

サーバの電源を切らない限り，たぶんIPアドレスは変わらないので，いざとなったらクライアント側の`/etc/hosts`を書き直してもらうことにした．現場にいれば，もう少しましな方法があったかもしれないが，何しろ相手はサンフランシスコにいるし，外部からログインできるような設定はされていない．本当はルータの設定で，IPアドレスの固定も外部からの接続もできるはずだが，こちらまで泥沼につかる必然性はない．そこまでいったらやはり，呪いか祟りということになってしまうだろう．

その後，Mさんは，仕事用のノートパソコンの環境を完全にLinuxに移行しようとし始めた．StarSuiteとWnn7を購入し，今までWindowsで作成したデータとの互換性をとり，WebはMozillaとなったところで，メーラを何にするかという問題が出てきたようだ．

次の電話は，なぜか夜の10時頃にかかってきた．どうしたのかと思ったら昨日，日本に来たという．EvolutionはOutlookみたいで嫌だし，Mozillaのメーラは使いにくいというのが，Mさんの意見である．自分で探してきたSylpheedを使い始めたようだが，日本語文書の禁則処理と，英語文書のスペルチェックが入っていないのが不便だそうである．だったら外部エディタを定義して，それを使用すればいいということになる．そうなると，エディタは当然Emacsということになる．

Emacsなら禁則処理はあるし，スペルチェックは`ispell`がEmacs内部から直接呼び出せる．今度はメールと電話で，Emacsと`.Xdefaults`の設定について，やりとりが始まった．夜10時になると電話がかかってきて，11頃になると「今日はもう寝る」で電話が終わるというパターンが，ここのところ続いている．今はSylpheedを使用しているようだが，メーラをMewに移行させる計画が，徐々に進行中である．

あとはRedHat8.0でよく話題になるフォントの設定を少し変更すれば，Mさんの要求はほぼ満たしそうだ．Mさんは，これだけいろいろなことをさせられるLinuxは，自分のような老人のボケ防止によさそうだとかいっているが，まさかMさんが50代も半ばになって，Linuxを本格的に使い始めるとは思わなかった．時代も変わったものである．

すけやす・しげお　インターメディア・アクセス

読者の広場

2003年3月号特集「ICカード技術の基礎と応用」に関して

▷ICカードにOSが搭載されていることを今回の特集で初めて知りました．OSによりアプリケーション開発の敷居が下がれば，まったく新しいキラーアプリケーションにより，ビジネスチャンスが広がりそうです．　(moto)

▷最近よく見かけるようになったICカードですが，磁気カードと大差ない技術かと思っていただけに，その内部に触れることができ，興味深く読みました．実際の開発も，普通の組み込み機器開発と同じように行えるため，比較的簡単に開発できそうです．　(た)

［編］ここ最近のICカードの普及にはめざましいものがあります．そのわりには情報が少なく，謎の存在だったのではないかと思うのですが，こうして特集として取り上げることができ，良かったと思います．

▷ICカード上でJavaが動作しているということには驚いた．これなら(Javaに精通している人なら)すぐにでもアプリケーションが開発できそうだ．もう一つ盛り上がりに欠ける組み込みJavaだが，こういった方面から普及し始めるのではないだろうか．　(emanon)

［編］携帯電話などでは使われているJavaですが，ICカードのような資源に強い制約のある環境でも動作できるという点に時代の変化を感じます．さらに，技術者のノウハウの流用が可能という点でも，Javaが動作するというのは魅力でしょう．

▷長年にわたってファームウェア技術と関わってきて，ここ最近はオープン系ソフトウェアに関わり，I/F誌の取り上げる技術とは少し遠いところにいたのですが，今回のICカード技術の特集は身近で，技術者ではない読者にも楽しめたと思います．今後もI/F誌を購読し続けたいと思います．　(プロテクトX)

▷最近，ICカード分野にもITRONが進出してきたというのには驚きました．たしかに組み込み系では大きなシェアを占めるITRONですが，ICカードでも同じような開発環境が使えるといいですよね．　(青海)

その他

▷「シニアエンジニア…」のゴキブリの話は面白かった．私の工場＆事務所にはネズミが出没します．小さなハムスターのようなネズミです．それがパソコン内に入って糞をたっぷり残してくれる．それを発見してからは，もう社内のパソコンの修理・改造をやめて，使い切りになりました．乾いた糞がファンの風で排出されていないかと戦々恐々．排気ファンの風を人に向けてはいけません．　(はくりゅう)

▷「外部メディアのバックアッププログラムを作成する」は，最近のI/F誌の傾向からすると多少色が違うような気もしたが，多くの分野でメディアが利用されるようになった現状を考えると「気の利いたツール」で参考になった．時折でいいので，今回みたいな良質なツールも紹介していただければと思います．　(の)

▷現在，元気のない日本にとって組み込み系「ユビキタスコンピューティング」のさまざまなアーキテクチャ(コア技術，ハードウェア周辺技術，OS，アプリケーションなど)が世界標準となり，全世界をリードすることを夢見ています．　(白石隆)

▷「フリーソフトウェア徹底活用講座」は，実例をあげてソースと出力を対比しながら進んでいくのでたいへん参考になります．コンパイラの最適化パターンを頭の中に叩き込んでコーディングしていくというのは，PC系では10年くらい前までは常識だったのですが，組み込み系では今でも必須なのですね……．　(Tr)

特集担当デスクから

☆組み込み機器開発の現場が「つらく」なってきている，という話をときどき聞くことがあります．今回の特集は，実際の現状や，課題がどこにあるかを探らなくては，という気持ちが出発点でした．

☆今回のような，開発現場エンジニアの立場とコンサルタントの立場，という対比は小誌でもはじめてだと思います．コンサルタントというと，現場エンジニアからはとっつきにくい印象を受けがちですが，特集執筆陣の「組み込み開発に注力するコンサルタント」の方々の多くは，現場エンジニアとしての経験も積みながら，コンサルティング業務を進めている方々です．抽象的な話題が開発現場になかなかなじまない状況もありますが，ある程度「長い目」で自分の仕事を見つめ直すのも必要ではないでしょうか？

☆前半の現場エンジニアの視点からの解説は，2001年3月の特集「オブジェクト指向の本格活用テクニック」第1章～第4章の発展形で，今回はより具体的なエレベータ/電波時計のモデルが解説され，実装コードも付属CD-ROMに収録できました．

☆組み込み関連で「つらい」要素ばかりではないとも思います．たとえば無線LANやカーエレクトロニクス，組み込みLinuxなど，いろいろ展開がはかれそうな分野もあるのではないでしょうか？

読者の広場

アンケートの結果

興味のあった記事（2003年3月号で実施）

①プロローグ カード社会とICカードの必要性
②第1章 ICカードOS「MULTOS」によるカードアプリケーションの作成
③第3章 非接触ICカード技術「FeliCa」の概要
④第2章 ICカードOS「ASEPcos」での開発とセキュリティ
⑤第5章 JavaCardの開発とメーラシステムへの応用
⑤第6章 次世代スマートカードの技術と応用
⑦第4章 eTRONの概要
⑧フリーソフトウェア徹底活用講座（第7回）
⑨フジワラヒロタツの現場検証（第68回）
⑩組み込みプログラミングノウハウ入門（第9回）
⑪gdb+DDDによるGUI対応デバッグ環境の構築
⑫開発技術者のためのアセンブラ入門（第16回）
⑬プログラミングの要（第1回）
⑬PCIデバイス対応デバイスドライバの作成法
⑮音楽配信技術の最新動向（第2回）
⑮ステレオオーディオDSPプログラミング入門（前編）
⑰開発環境探訪（第16回）
⑱リアルタイムOS「INTEGRITY」の概要
⑲Internet World Asia 2002
⑲ハッカーの常識的見聞録（第27回）

特集「ICカード技術の基礎と応用」についてのアンケートの結果

Q1　ICカードのアプリケーションとして，どのようなものを期待していますか？（複数回答可）

①ポイントカード（46%）
②プリペイドカード（8%）
③入場チケット（8%）
④セキュリティカード（69%）
⑤その他（16%）

虹彩や指紋などとの2重チェックが必要．ICカードでお金を扱いたくない．

Q2　ICカードに関して，興味をもっている内容は？（複数回答可）

①規格（31%）
②ハードウェア（23%）
③OS（31%）
④アプリケーション（38%）
⑤セキュリティ（38%）
⑥その他（15%）

多種アプリケーションの自由な搭載．アプリケーションインターフェースと標準化．

Q3　あなたはICカードを何枚もっていますか？

0枚（58%）
1枚（0%）
2枚（8%）
3枚（25%）
4枚（0%）
5枚（8%）

次号予告 TCP/IPの現在とVoIP技術の全貌

2003年6月号は4月25日発売です

TCP/IPの基礎と現状/VoIPの基礎/シグナリング/CODEC/ネットワーク構築/製品事例/LinuxにおけるVoIPの実装

インターネットのインフラとして使われているTCP/IPが誕生してから20年が過ぎた．基本的な部分こそ20年前から変化していないが，その細部を見ると確実にバージョンアップが行われている．そこで本特集では，TCP/IPの基礎と，TCP/IPに関する近年のトピックスなどをまとめて紹介する．

また，TCP/IPの応用例の一つとして，昨今のブロードバンド環境の普及にともない，インターネット電話が注目されている．その根幹技術であるVoIPは，音声のディジタル化とエンコード，ディレクトリサーバによる通信対象の探索，インターネットにおけるQoSの維持，NATなど，ネットワーク技術の集大成ともいえるものだ．

そこで本特集ではVoIP技術について，VoIPプロトコルスタックから音声CODEC，ネットワークの構築にいたるまでを徹底解説する．

編集後記

■他人事さぁ，と思っていたInfluenza．下の娘(6才)が発熱→風邪かい?って医者にみせたら「判定：香港A型．薬で熱は下がるけれど，下がってから二日は外出しちゃいけません」ですと．参ったねっていってたら，約2日間隔で上の娘(8才)，妻と「飛び火」．同様の状況を「逃げ切った」友人にならい，私も罹らずにすませます．（洋）

■ふと，頭の中で曲が始まると，それが止まらなくなり，最後まで演奏してしまう……という経験は誰にでもあるはず(あるよね？ あるといって！)．最近，この症状が酷くなり，四六時中何かを演奏している．これを書いている最中にも，頭の中ではドアーズの「Break on through」が鳴っている．この症状を止める策って無い？（= IO）

■本誌付属CD-ROM InterGigaもついにNo.30になりました．No.1制作当時はまだまだWin3.1やメモリ8Mの486マシンが現役でしたが，今じゃ1枚512Mのメモリモジュールやらクロック3GHzなんてCPUが売られていますヨ．No.40や50になったら，いったいどんな環境になるんでしょうか(そこまでIGが続くようにがんばらねば^^;)（M）

■私が日本人の中で一番好きなミュージシャンは，風呂なしアパートに住んでいるんだそうです．いいものを作ることと儲かることが乖離している例ですが，音楽界に限った話ではないだけに他人事ではありません．この二つを結び付けるビジネスモデルを早急に作らねば．なんとかしなければと焦るばかり．（み）

■ちょっと前に，パウエル氏が不似合いな場所によく来たものだと思っていたら，ビル・ゲイツ氏までが同じようなスタイルでとても不似合いな場所にやってきた．二人は互いに異なる世界にいる人物であり，来日した目的も違うのだが，なぜか同じような下心で来たような印象を受ける．また，議員の対応の仕方と考え方まで同じに見えるのだが．（Y）

■都内ではまだ花粉があまり飛んでいない頃に伊豆へ行ったのですが，暫くすると花粉症の症状が出始めました．よくよく周りを見てみると，杉が道路の両端に植えてあるではないですか．折角の観光気分もなんだかへこみ気味．もう都内でも花粉症の人は苦しんでいるでしょうが，この時期は旅行先の花粉情報も要チェックですよ！（Y2）

■グローバリズムが破綻した後のアメリカはデモクラシーを大義名分にした．すなわち民主主義は正義であり，独裁国家を「悪の枢軸」と呼び，歯向かう者に時には戦争のテクノロジーで叩くという構造だ．自分達を正義だと言った瞬間にリアリズムが非常に見えにくくなる気がする．（ちゃん）

■今，学校でウンチをしない小学生が4割もいる！ これが問題になっているらしい．理由は個室に入るだけでイジメられるから等々．でもそれって普通ですよね．十数年前私が小学生の時も同じでした．子供にだって自意識はあるし，みんな単純な下ネタが好きだったし，教室で，1の川と2の川の間に落ちていたこともあったな．（ヨ）

お知らせ

▶読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

▶投稿歓迎

本誌に投稿をご希望の方は，連絡先(自宅/勤務先)を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です(送り先はsupportinter@cqpub.co.jpまで)．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

▶本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版(株)の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

▶コピーサービスのご案内

本誌バックナンバーの掲載記事については，在庫(原則として24か月分)のないものに限りコピーサービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

- コピー料金(税込み)
  1ページにつき100円
- 発送手数料(判型に関わらず)
  1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円
- 送付金額の算出方法
  総ページ数×100円＋発送手数料
- 入金方法
  現金書留か郵便小為替による郵送
- 明記事項
  雑誌名，年月号，記事タイトル，開始ページ，総ページ数
- 宛て先
  〒170-8461 東京都豊島区巣鴨1-14-2
  CQ出版株式会社 コピーサービス係
  (TEL：03-5395-4211，FAX：03-5395-1642)

▶お問い合わせ先のご案内

- 在庫，バックナンバー，年間購読送付先変更に関して
  販売部：03-5395-2141
- 広告に関して
  広告部：03-5395-2133
- 雑誌本文に関して
  編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

©CQ出版(株) 2003 振替 00100-7-10665
2003年5月号 第29巻 第5号(通巻第311号)
2003年5月1日発行(毎月1日発行)
定価は裏表紙に表示してあります

発行人／蒲生良治
編集人／相原 洋
編集／大野典宏 村上真紀 山口光樹 小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／森 祐子 唐沢睦子
広告／澤辺 彰 中元正夫 渡部真美

発行所／CQ出版株式会社 〒170-8461 東京都豊島区巣鴨1-14-2
電話／編集部(03)5395-2122 URL http://www.cqpub.co.jp/interface/
広告部(03)5395-2133 インターフェース編集部へのメール
販売部(03)5395-2141 supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社 美和印刷株式会社
製本／星野製本株式会社

日本ABC協会加盟誌
(新聞雑誌部数公査機構) ISSN0387-9569

Printed in Japan